

IEEE P1003.2 Draft 11.2 – September 1991

Copyright © 1991 by the
Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, NY 10017, USA
All rights reserved as an unpublished work.

This is an unapproved and unpublished IEEE Standards Draft, subject to change. The publication, distribution, or copying of this draft, as well as all derivative works based on this draft, is expressly prohibited except as set forth below.

Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities only, and subject to the restrictions contained herein.

Permission is hereby also granted for member bodies and technical committees of ISO and IEC to reproduce this document for purposes of developing a national position, subject to the restrictions contained herein.

Permission is hereby also granted to the preceding entities to make limited copies of this document in an electronic form only for the stated activities.

The following restrictions apply to reproducing or transmitting the document in any form: 1) all copies or portions thereof must identify the document's IEEE project number and draft number, and must be accompanied by this entire notice in a prominent location; 2) no portion of this document may be redistributed in any modified or abridged form without the prior approval of the IEEE Standards Department.

Other entities seeking permission to reproduce this document, or any portion thereof, for standardization or other activities, must contact the IEEE Standards Department for the appropriate license.

Use of information contained in this unapproved draft is at your own risk.

IEEE Standards Department
Copyright and Permissions
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA
+1 (908) 562-3800
+1 (908) 562-1571 [FAX]

STANDARDS PROJECT

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

Part 2: Shell and Utilities

Sponsor
**Technical Committee on Operating Systems
and Application Environments**
of the
IEEE Computer Society

Work Item Number: JTC 1.22.21.2

Abstract: ISO/IEC 9945-2: 199x (IEEE Std 1003.2-199x) is part of the POSIX series of standards for applications and user interfaces to open systems. It defines the applications interface to a shell command language and a set of utility programs for complex data manipulation.

Keywords: API, application portability, data processing, open systems, operating system, portable application, POSIX, shell and utilities

P1003.2 / D11.2

September 1991

This is an unapproved IEEE Standards Draft, subject to change. Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities. Permission is also granted for member bodies and technical committees of ISO and IEC to reproduce this document for purposes of developing a national position.

Other entities seeking permission to reproduce this document for standardization or other activities, or to reproduce portions of this document for these or other uses, must contact the IEEE Standards Department for the appropriate license. Use of information contained in this unapproved draft is at your own risk.

IEEE Standards Department
Copyright and Permissions
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA
+1 (908) 562-3800
+1 (908) 562-1571 [FAX]
September 1991

SH XXXXX

Editor's Notes

The IEEE ballot for Draft 11.2 is due at the IEEE Standards Office on **21 October 1991**. You are also asked to e-mail any balloting comments to me: hlj@posix.com. Please read the balloting instructions in Annex G.

This document is also registered as ISO/IEC CD 9945-2.2. The international balloting period is unrelated to the IEEE balloting. Member bodies, please consult any accompanying materials from SC22. Also, please read the remainder of these Editor Notes to see explanations of stylistic differences between a draft and the final standard (copyright notices, inline rationale, etc.).

The IEEE balloting will be on hiatus during the international balloting period, which is probably scheduled to complete at the May 1992 WG15 meeting. This is in accordance with the WG15 Synchronization Plan, which calls for coordinated balloting to result in the approval of an IEEE/ANSI standard that is identical to the ISO/IEC Draft International Standard (DIS). There will be a final recirculation of a full draft (12) to the IEEE balloting group before it is sent to the Standards Board.

This section will not appear in the final document. It is used for editorial comments concerning this draft. Draft 11.2 is the fifth recirculation of the balloting process that began in December 1988 with Draft 8. Please consult Annex G and the cover letter for the ballot that accompanied this draft for information on how the recirculation is accomplished.

This draft uses small numbers in the right margin in lieu of change bars. “2” denotes changes from Draft 11.1 to Draft 11.2. “1” denotes changes from Draft 11 to Draft 11.1. All diff-marks prior to Draft 11.1 have been removed. Trivial informative (i.e., non-normative) changes and purely editorial changes such as grammar, spelling, or cross references are not diff-marked.

There are two versions of Draft 11.2 in circulation. The full printed version was sent for SC22 balloting and is also available from the IEEE for a duplication fee [call (800) 678-IEEE or +1 (908) 981-1393 outside the US]. The version sent to the IEEE balloting group consists (mostly) of pages containing normative changes. This was done to focus balloting group attention on the changes being balloted and to reduce costs and administrative time. The changes-only version contains a few handwritten pointers in the margins to show context where it would not be obvious; numbers near the normal page numbers show what the corresponding Draft 11 page number would be.

The following minor global changes have been made without diff-marks:

- Instances of the verbs “print,” “report,” “display,” “issue,” and “list” are being changed to “write” as part of a general cleanup related to the UPE, where “write” and “display” have precise meanings. This is probably not completed and will continue throughout ballot resolution and the final editing process.

ISO and IEEE have tightened up the requirements for the use of “shall.” We have been directed that all sentences that are currently declarative must be changed to use the “shall” form if they pose a requirement: “The status is zero” → “The status shall be zero.” One specific instance of this was changing “The following options/operands are available” to “The following options/operands shall be supported by the implementation.” Another: “The `foo` utility follows the utility argument syntax standard described in 2.11.2” to “The `foo` utility shall conform to the utility argument syntax guidelines described in 2.10.2.” It is a tedious process to do all these translations and they are not complete. They will be completed on a draft-by-draft basis. In the meantime, please assume that all declarative sentences mean to use “shall” and treat them as either implementation or application requirements unless they specifically say “may,” “should,” or “can.”

The rationale text for all the sections has been temporarily moved from Annex E and interspersed with the appropriate sections. The rationale sections are identified with the phrase “*(This subclause is not a part of P1003.2)*” in the heading. This collocation of rationale with its accompanying text was done to encourage the Technical Reviewers to maintain the rationale text, as well as provide explanations to the reviewers and balloters. Not all of the Rationale sections have contents as of this draft. The empty sections may be partially distracting, but we feel it is imperative to keep them there to encourage the Technical Reviewers to provide rationale as needed.

Please report typographical errors to:

Hal Jespersen
POSIX Software Group
447 Lakeview Way
Redwood City, CA 94062
+1 (415) 364-3410
FAX: +1 (415) 364-4498
Email: hlj@Posix.COM

(Electronic mail is preferred.)

The copying and distribution of IEEE balloting drafts is accomplished by the Standards Office. To report problems with reproduction of your copy, contact:

Anna Kaczmarek
IEEE Standards Office
P.O. Box 1331
445 Hoes Lane
Piscataway, NJ 08855-1331
+1 (908) 562-3811
FAX: +1 (908) 562-1571

2

2

2

82	Additional copies of this draft are available for a duplication and mailing fee.	2
83	Contact:	2
84	IEEE Publications	2
85	1 (800) 678-IEEE	2
86	+1 (908) 981-1393 [outside US]	2
87	This draft is available in various electronic forms to assist the review process.	2
88	Our thanks to Andrew Hume of AT&T Bell Laboratories for providing online	2
89	access facilities. Note that this is a limited experiment in providing online access;	2
90	future ballots may provide other forms, such as diskettes or a bulletin board	2
91	arrangement, but the instructions shown here are the only methods currently	2
92	available. Please also observe the additional copyright restrictions that are	2
93	described in the online files.	2
94	Assuming you have access to the Internet, the scenario is approximately	2
95	ftp research.att.com # research's IP address is 192.20.225.2	2
96	<login as netlib; password is your email address>	2
97	cd posix/p1003.2/d11.2	2
98	get toc index	2
99	binary	2
100	get p11-20.Z	2
101	The draft is available in several forms. The table of contents can be found in <code>toc</code> ,	2
102	pages containing a particular section are stored under the section number, sets of	2
103	pages are stored in files with names of the form <code>pn-m</code> , and the entire draft is	2
104	stored in <code>all</code> . By default, files are ASCII. A <code>.ps</code> suffix indicates PostScript. A <code>.Z</code>	2
105	suffix indicates a compress'ed file. The file <code>index</code> contains a general description	2
106	of the files available.	2
107	These files are also available via electronic mail by sending a message like	2
108	send 3.4 3.5 9.2 from posix/p1003.2/d11.2	2
109	to netlib@research.att.com. If you use email, you should <i>not</i> ask for the	2
110	compressed version. For a more complete introduction to this form of <i>netlib</i> , send	2
111	the message	2
112	send help	2

POSIX.2 Change History

This section is provided to track major changes between drafts. Since it was first added in Draft 11, earlier entries omit some degree of detail.

- | | | |
|------------|---|---|
| Draft 11.2 | [September 1991] Sixth IEEE ballot (fifth recirculation; only changed pages distributed). Second ISO/IEC CD 9945-2 registration (full draft distributed). | 2 |
| | — Equivalence classes as starting/ending points of regular expression bracket expression range expression have been made unspecified. | 2 |
| | — The LC_COLLATE substitute keyword has been deleted. | 2 |
| | — cksum (4.9): Modifications to the algorithm. | 2 |
| | — cp (4.13): Restoration of the | 2 |
| | — stty (4.59): Addition of the tostop operand. | 2 |
| | — lex (A.2): Further clarification of ERE differences. | 2 |
| | — Miscellaneous clarifications to various utilities. | 2 |
| Draft 11.1 | [June 1991] Fifth IEEE ballot (fourth recirculation; only changed pages distributed). | 1 |
| | — Modification of the definition of <i>byte</i> and clarifications of octal/hexadecimal byte representations throughout the utilities. | 1 |
| | — Clarifications to the locale definition source file description in 2.5; addition of a yacc grammar. | 1 |
| | — Removal of pax -e character translation option. | 1 |
| | — Miscellaneous clarifications to various utilities. | 1 |
| | — Reconciliation of feature test macros and headers in Annex B with POSIX.1. | 1 |
| Draft 11 | [February 1991] Fourth IEEE ballot (third recirculation). | |
| | — Changes in 2.3 to the treatment of regular built-ins in regards to their <i>exec</i> -able versions. | |
| | — Changes to 2.4 (character names and charmap syntax) and 2.5 (localedef input format) as a result of international balloting. Addition of the {POSIX2_LOCALEDEF} symbol. | |
| | — Changes to the shell quoting rules, arithmetic expression syntax, command search order, error descriptions, and exportable functions. | |
| | — Movement of the <code>command</code> utility from special built-in status to be a utility in Section 4. | |

- `cp` (4.13): Significant clarifications and interface changes.
 - `date` (4.15): Added field descriptor modifiers to handle alternate calendar forms when supported by the locale and implementation.
 - `pax` (4.48): Significant interface changes, including international character set translations.
 - `test` (4.62): Deprecated some functionality due to inconsistent behavior in existing implementations that cause portability problems in existing applications.
 - `make` (6.2): Addition of the `.POSIX` special target, return of some rules to strict existing practice.
 - Miscellaneous clarifications to various utilities.
 - The FORTRAN section now has two options associated with it: Development Utilities (`fort77`) and Runtime Utilities (`asa`).
 - Addition of full example profiles and charmaps from Denmark in Annex F.
- Draft 10 [July 1990] Third IEEE ballot (second recirculation).
- This draft primarily has been one of clarification and amplification. In resolving ballot objections, large portions of the draft have been rewritten, affecting all sections, but comparatively few changes in [intended] functionality have occurred.
 - New shell command language features (see Section 3):
 - Utility name changes:
- | | <u>Draft 9</u> | <u>Draft 10</u> |
|--|----------------------|----------------------|
| | <code>create</code> | <code>pathchk</code> |
| | <code>hexdump</code> | <code>od</code> |
| | <code>sendto</code> | <code>mailx</code> |
- A few of the utilities and global sections now have a more formal description, using a `yacc`-like grammar.
 - Considerably more detail has been added to the internationalization features of the standard: global changes to clauses 2.4 and 2.5; new detail to the `LC_*` variables in each utility section; specification of **LC_MESSAGES** (replacing **LC_RESPONSE**).
 - Due to some ISO requirements, Sections 1 and 2 have been reorganized yet again, causing many cross reference number changes. The Related Standards annex has been turned into simply a Bibliography. The Non-Specified Language Compilers annex has been replaced by a Sample National Profile

190 annex.

191 Draft 9 [August 1989] Second IEEE ballot (first recirculation). Also
192 registered as ISO/IEC CD 9945-2.1. A few minor corrections to
193 some sections. :-)

194 Draft 8 [December 1988] First IEEE ballot. Also submitted to ISO/IEC
195 JTC 1/SC22 for review and comment.

196 Draft 7 [September 1988] “Mock ballot” conducted by working group
197 members only.

198 *POSIX.2 Technical Reviewers*

199 The individuals denoted in Table i are the Technical Reviewers for this draft.
200 During balloting they are the subject matter experts who coordinate the resolu-
201 tion process for specific sections, as shown.

202 **Table i — POSIX.2 Technical Reviewers**

203

Section		Description	Reviewer	
205	1	<i>General</i>	Jespersen	
206	2.4,2.5	<i>Definitions (Locales)</i>	Leijonhufvud	1
207	2 (rest)	<i>Definitions (Various)</i>	Jespersen	
208	3	<i>Command Language</i>	Jespersen	
209	4	<i>Execution Environment Utilities: cp, rm</i>	Bostic	2
210	4	<i>Execution Environment Utilities: (the rest)</i>	Jespersen	2
211	6	<i>Software Development Utilities</i>	Jespersen	
212	7	<i>Language-Independent Bindings</i>	Jespersen	2
213	A	<i>C Development Utilities</i>	Jespersen	
214	B	<i>C Bindings</i>	Jespersen	2
215	C	<i>FORTRAN Development and Runtime Utilities</i>	Jespersen	
216	D-G	<i>Various</i>	Jespersen	

217

218 Also, our special thanks to Donn Terry for writing or improving all the yacc-
219 based grammars used in Draft 10.

POSIX.2 Proposed Schedule

This section will not appear in the final document. It is used to provide editorial notes regarding the proposed POSIX.2 schedule. In the schedule, the UPE stands for “User Portability Extension.”

Date	Milestone (End of Meeting)	Draft
Sep 7-11, 1987 Nashua, NH	Utility format frozen; 10% of utilities described.	3
Dec 7-14, 87 San Diego, CA	50% of utilities described; shell update; substantial progress in Sections 2, 3, 4, 8.	4
Mar 14-18, 1988 Washington, DC	Utility selection frozen; 75% described.	5
Jul 11-15, 1988 Denver, CO	100% utilities described; functional freeze; produce “mock ballot” and POSIX FIPS draft 7	6
[Sep-Oct 1988]	[Mock ballot]	7
Oct 24-28, 1988 Honolulu, HI	Resolve mock ballot objections; produce first real ballot (draft 8) UPE planning begins	7
[Jan-Feb 1989]	[First ballot]	8
Jan 9-11, 1989 Ft. Lauderdale, FL	Begin UPE definitions; Technical Reviewer coordination of first ballot responses	8
[Feb-Apr 1989]	[Ballot resolution]	8
Apr 24-28, 1989 Minneapolis, MN	Working Group concurrence with ballot resolution; produce Draft 9 for recirculation; UPE work	9
Jul 10-14, 1989 San Jose, CA	UPE work	
[Oct 1989]	[First Recirculation]	9
[Nov-Feb 1990]	[Ballot resolution]	9
[Aug-Sep 1990]	[Second Recirculation]	10
[Mar 1991]	[Third Recirculation]	11
[Jun 1991]	[Fourth Recirculation]	11.1
[Sep 1991]	[Fifth Recirculation]	11.2
[mid-1992]	[IEEE Standard Board Approves??]	12
[Jul 1990 - Apr 1992]	[Ballot .2a UPE supplement]	

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. 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.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

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 all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, the IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

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

IEEE Standards documents are adopted by the Institute of Electrical and Electronics Engineers without regard to whether their adoption may involve patents on articles, materials, or processes. Such adoption does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the standards documents.

Contents

	PAGE
Introduction	viii
Organization of the Standard	viii
Base Documents	ix
Related Standards Activities	x
Section 1: General	1
1.1 Scope	1
1.2 Normative References	12
1.3 Conformance	12
Section 2: Terminology and General Requirements	19
2.1 Conventions	19
2.2 Definitions	23
2.3 Built-in Utilities	51
2.4 Character Set	54
2.5 Locale	61
2.6 Environment Variables	103
2.7 Required Files	109
2.8 Regular Expression Notation	110
2.9 Dependencies on Other Standards	138
2.10 Utility Conventions	147
2.11 Utility Description Defaults	156
2.12 File Format Notation	168
2.13 Configuration Values	173
Section 3: Shell Command Language	181
3.1 Shell Definitions	183
3.2 Quoting	185
3.3 Token Recognition	188
3.4 Reserved Words	190
3.5 Parameters and Variables	192
3.6 Word Expansions	195
3.7 Redirection	209
3.8 Exit Status and Errors	214
3.9 Shell Commands	216
3.10 Shell Grammar	233
3.11 Signals and Error Handling	240
3.12 Shell Execution Environment	240
3.13 Pattern Matching Notation	242
3.14 Special Built-in Utilities	246

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

	PAGE
Section 4: Execution Environment Utilities	263
4.1 awk — Pattern scanning and processing language	263
4.2 basename — Return nondirectory portion of pathname	297
4.3 bc — Arbitrary-precision arithmetic language	301
4.4 cat — Concatenate and print files	318
4.5 cd — Change working directory	322
4.6 chgrp — Change file group ownership	326
4.7 chmod — Change file modes	329
4.8 chown — Change file ownership	337
4.9 cksum — Write file checksums and sizes	341
4.10 cmp — Compare two files	347
4.11 comm — Select or reject lines common to two files	350
4.12 command — Execute a simple command	354
4.13 cp — Copy files	359
4.14 cut — Cut out selected fields of each line of a file	368
4.15 date — Write the date and time	373
4.16 dd — Convert and copy a file	379
4.17 diff — Compare two files	388
4.18 dirname — Return directory portion of pathname	395
4.19 echo — Write arguments to standard output	399
4.20 ed — Edit text	402
4.21 env — Set environment for command invocation	419
4.22 expr — Evaluate arguments as an expression	423
4.23 false — Return false value	428
4.24 find — Find files	430
4.25 fold — Fold lines	438
4.26 getconf — Get configuration values	442
4.27 getopts — Parse utility options	447
4.28 grep — File pattern searcher	452
4.29 head — Copy the first part of files	459
4.30 id — Return user identity	462
4.31 join — Relational database operator	466
4.32 kill — Terminate or signal processes	471
4.33 ln — Link files	476
4.34 locale — Get locale-specific information	480
4.35 localedef — Define locale environment	486
4.36 logger — Log messages	491
4.37 logname — Return user's login name	494
4.38 lp — Send files to a printer	496
4.39 ls — List directory contents	502
4.40 mailx — Process messages	510
4.41 mkdir — Make directories	514
4.42 mkfifo — Make FIFO special files	518
4.43 mv — Move files	521
4.44 nohup — Invoke a utility immune to hangups	526
4.45 od — Dump files in various formats	530
4.46 paste — Merge corresponding or subsequent lines of files	538

	PAGE
4.47 pathchk — Check pathnames	543
4.48 pax — Portable archive interchange	548
4.49 pr — Print files	562
4.50 printf — Write formatted output	568
4.51 pwd — Return working directory name	574
4.52 read — Read a line from standard input	576
4.53 rm — Remove directory entries	579
4.54 rmdir — Remove directories	584
4.55 sed — Stream editor	587
4.56 sh — Shell, the standard command language interpreter	597
4.57 sleep — Suspend execution for an interval	603
4.58 sort — Sort, merge, or sequence check text files	605
4.59 stty — Set the options for a terminal	613
4.60 tail — Copy the last part of a file	623
4.61 tee — Duplicate standard input	628
4.62 test — Evaluate expression	631
4.63 touch — Change file access and modification times	640
4.64 tr — Translate characters	645
4.65 true — Return true value	652
4.66 tty — Return user's terminal name	654
4.67 umask — Get or set the file mode creation mask	657
4.68 uname — Return system name	662
4.69 uniq — Report or filter out repeated lines in a file	665
4.70 wait — Await process completion	669
4.71 wc — Word, line, and byte count	674
4.72 xargs — Construct argument list(s) and invoke utility	678
Section 5: User Portability Utilities Option	685
Section 6: Software Development Utilities Option	687
6.1 ar — Create and maintain library archives	687
6.2 make — Maintain, update, and regenerate groups of programs	695
6.3 strip — Remove unnecessary information from executable files	716
Section 7: Language-Independent System Services	719
7.1 Shell Command Interface	720
7.2 Access Environment Variables	720
7.3 Regular Expression Matching	721
7.4 Pattern Matching	721
7.5 Command Option Parsing	721
7.6 Generate Pathnames Matching a Pattern	722
7.7 Perform Word Expansions	722
7.8 Get POSIX Configurable Variables	722
7.9 Locale Control	723
Annex A (normative) C Language Development Utilities Option	725
A.1 c89 — Compile Standard C programs	726

	PAGE
A.2 lex — Generate programs for lexical tasks	736
A.3 yacc — Yet another compiler compiler	750
Annex B (normative) C Language Bindings Option	771
B.1 C Language Definitions	772
B.1.1 POSIX Symbols	772
B.1.2 Headers and Function Prototypes	774
B.1.3 Error Numbers	774
B.2 C Numerical Limits	775
B.2.1 C Macros for Symbolic Limits	775
B.2.2 Compile-Time Symbolic Constants for Portability Specifications	776
B.2.3 Execution-Time Symbolic Constants for Portability Specifications	777
B.2.4 POSIX.1 C Numerical Limits	777
B.3 C Binding for Shell Command Interface	778
B.3.1 C Binding for Execute Command	778
B.3.2 C Binding for Pipe Communications with Programs	782
B.4 C Binding for Access Environment Variables	786
B.5 C Binding for Regular Expression Matching	786
B.6 C Binding for Match Filename or Pathname	794
B.7 C Binding for Command Option Parsing	796
B.8 C Binding for Generate Pathnames Matching a Pattern	799
B.9 C Binding for Perform Word Expansions	804
B.10 C Binding for Get POSIX Configurable Variables	809
B.11 C Binding for Locale Control	812
Annex C (normative) FORTRAN Development and Runtime Utilities	
Options	813
C.1 asa — Interpret carriage-control characters	813
C.2 fort77 — FORTRAN compiler	817
Annex D (informative) Bibliography	825
Annex E (informative) Rationale and Notes	829
E.1 General	829
E.2 Terminology and General Requirements	830
E.3 Shell Command Language	831
E.4 Execution Environment Utilities	832
E.5 User Portability Utilities Option	843
E.6 Software Development Utilities Option	843
E.7 Language-Independent System Services	844
E.8 C Language Development Utilities Option	844
E.9 C Language Bindings Option	845
E.10 FORTRAN Development and Runtime Utilities Options	846
Annex F (informative) Sample National Profile	847

	PAGE
Annex G (informative) Balloting Instructions	919
Identifier Index	929
Alphabetic Topical Index	933

FIGURES

Figure B-1 – Sample system() Implementation	781
Figure B-2 – Sample pclose() Implementation	785
Figure B-3 – Example Regular Expression Matching	791
Figure B-4 – Argument Processing with getopt()	798

TABLES

Table 2-1 – Typographical Conventions	19
Table 2-2 – Regular Built-in Utilities	51
Table 2-3 – Character Set and Symbolic Names	54
Table 2-4 – Control Character Set	56
Table 2-5 – LC_CTYPE Category Definition in the POSIX Locale . . .	68
Table 2-6 – Valid Character Class Combinations	71
Table 2-7 – LC_COLLATE Category Definition in the POSIX Locale . .	74
Table 2-8 – LC_MONETARY Category Definition in the POSIX Locale .	84
Table 2-9 – LC_NUMERIC Category Definition in the POSIX Locale .	88
Table 2-10 – LC_TIME Category Definition in the POSIX Locale . . .	89
Table 2-11 – LC_MESSAGES Category Definition in the POSIX Locale .	92
Table 2-12 – BRE Precedence	117
Table 2-13 – ERE Precedence	120
Table 2-14 – C Standard Operators and Functions	146
Table 2-15 – Escape Sequences	169
Table 2-16 – Utility Limit Minimum Values	173
Table 2-17 – Symbolic Utility Limits	174
Table 2-18 – Optional Facility Configuration Values	179
Table 4-1 – awk Expressions in Decreasing Precedence	268
Table 4-2 – awk Escape Sequences	289
Table 4-3 – bc Operators	307
Table 4-4 – ASCII to EBCDIC Conversion	385
Table 4-5 – ASCII to IBM EBCDIC Conversion	386
Table 4-6 – dirname Examples	398
Table 4-7 – expr Expressions	425
Table 4-8 – od Named Characters	534
Table 4-9 – stty Control Character Names	617

Table 4-10	–	stty Circumflex Control Characters	618
Table 7-1	–	POSIX.1 Numeric-Valued Configurable Variables	723
Table A-1	–	lex Table Size Declarations	741
Table A-2	–	lex Escape Sequences	743
Table A-3	–	lex ERE Precedence	743
Table A-4	–	yacc Internal Limits	765
Table B-1	–	POSIX.2 Reserved Header Symbols	773
Table B-2	–	_POSIX_C_SOURCE	773
Table B-3	–	C Macros for Symbolic Limits	775
Table B-4	–	C Compile-Time Symbolic Constants	776
Table B-5	–	C Execution-Time Symbolic Constants	777
Table B-6	–	Structure Type regex_t	787
Table B-7	–	Structure Type regmatch_t	787
Table B-8	–	regcomp() cflags Argument	787
Table B-9	–	regexexec() eflags Argument	787
Table B-10	–	regcomp() , regexexec() Return Values	790
Table B-11	–	fnmatch() flags Argument	794
Table B-12	–	Structure Type glob_t	800
Table B-13	–	glob() flags Argument	800
Table B-14	–	glob() Error Return Values	802
Table B-15	–	Structure Type wordexp_t	804
Table B-16	–	wordexp() flags Argument	805
Table B-17	–	wordexp() Return Values	806
Table B-18	–	confstr() name Values	809
Table B-19	–	C Bindings for Numeric-Valued Configurable Variables	811

Introduction

(This Introduction is not a normative part of P1003.2 Information technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities, but is included for information only.)

The purpose of this standard is to define a standard interface and environment for application programs that require the services of a “shell” command language interpreter and a set of common utility programs. It is intended for systems implementors and application software developers, and is complementary to ISO/IEC 9945-1: 1990 {8} (first in a family of “POSIX” standards), which specifies operating system interfaces and source code level functions, based on the UNIX¹⁾ system documentation. This standard, or “POSIX.2,” is based upon documentation and the knowledge of existing programs that assume an interface and architecture similar to that described by POSIX.1. (See 1.1 for a full description of the relationship between the standards.)

The majority of this standard describes the functions of utilities that can interface with application programs. The standard also provides high-level language interfaces that the application uses to access these utilities and other useful, related services. These language-independent service interfaces are temporarily described in terms of their C language bindings. The C language assumed is that defined by the C Standard: *ANSI/X3.159-1989 Programming Language C Standard* produced by Technical Committee X3J11 of the Accredited Standards Committee X3 — Information Processing Systems.

Organization of the Standard

The standard is divided into ten parts:

- General, including a statement of scope, normative references, and conformance requirements. (Section 1).
- Definitions, general requirements, and the environment available to applications. (Section 2).
- The shell command interpreter language. (Section 3).
- Descriptions of the utilities in the required “Execution Environment Utilities.” (Section 4).
- Descriptions of the utilities required for user portability on asynchronous terminals. (Section 5 [to be provided in a future revision]).
- Descriptions of the utilities in the optional “Software Development Utilities.” (Section 6).

1) UNIX is a registered trademark of UNIX System Laboratories in the USA and other countries.

- Language-independent interfaces for high-level programming language access to shell and related services. (Section 7).
- Descriptions of the utilities in the optional “C Language Development Utilities.” (Normative Annex A).
- C language bindings to the interfaces in Section 6. (Normative Annex B).
- Descriptions of the utilities in the optional “FORTRAN Development and Runtime Utilities.” (Normative Annex C).

This introduction, the foreword, any footnotes, NOTES accompanying the text, and the *informative* annexes are not considered part of the standard. Annexes D through G are informative.

Base Documents

Many of the interfaces and utilities of this standard were adapted from materials in machine-readable forms donated by the following organizations:

- AT&T: the *System V Interface Definition (SVID)* {B24},²⁾ Issue 2, Volume 2. Copyright © 1986, AT&T; reprinted with permission.
- The X/Open Company, Ltd.: the *X/Open Portability Guide* {B30} {B31}, Issues II and III, Volume 1. Copyright © 1989, X/Open Company, Ltd; reprinted with permission.
- University of California, *The UNIX User's Reference Manual* {B28}, 4.3 Berkeley Software Distribution, Virtual VAX-11 Version, 1986. Copyright © 1980, 1983, The Regents of the University of California; reprinted with permission.³⁾

Significant reference use was also made of the following books:

- Bolsky, Morris I., Korn, David G., *The KornShell Command and Programming Language* {B25}, Prentice Hall, Englewood Cliffs, New Jersey (1988).
- Aho, Alfred V., Kernighan, Brian W., Weinberger, Peter J., *The AWK Programming Language* {B21}, Addison-Wesley, Reading, Massachusetts (1988).

Many other proposals for functions and utilities were received from the various working group members, who are listed in the Acknowledgements section of this standard.

2) The number in braces corresponds to those of the references in 1.2 (or the bibliographic entry in Annex D if the number is preceded by the letter B).

3) The IEEE is grateful to AT&T, UniForum, and the Regents of the University of California for permission to use their machine-readable materials.

Related Standards Activities

Activities to extend this standard to address additional requirements are in progress, and similar efforts can be anticipated in the future.

The following areas are under active consideration at this time, or are expected to become active in the near future:⁴⁾

- (1) Language-independent service descriptions of POSIX.1 {8}
- (2) C, Ada, and FORTRAN Language bindings to (1)
- (3) Verification testing methods
- (4) Realtime facilities
- (5) Secure/Trusted System considerations
- (6) Network interface facilities
- (7) System Administration
- (8) Graphical User Interfaces
- (9) Profiles describing application- or user-specific combinations of Open Systems standards for: supercomputing, multiprocessor, and batch extensions; transaction processing; realtime systems; and multiuser systems based on historical models
- (10) An overall guide to POSIX-based or related Open Systems standards and profiles

Extensions are approved as “amendments” or “revisions” to this document, following the IEEE and ISO/IEC Procedures.

Approved amendments are published separately until the full document is reprinted and such amendments are incorporated in their proper positions.

If you have interest in participating in the TCOS working groups addressing these issues, please send your name, address, and phone number to the Secretary, IEEE Standards Board, Institute of Electrical and Electronics Engineers, Inc., P.O. Box 1331, 445 Hoes Lane, Piscataway, NJ 08855-1331, and ask to have this forwarded to the chairperson of the appropriate TCOS working group. If you have interest in participating in this work at the international level, contact your ISO/IEC national body.

4) A *Standards Status Report* that lists all current IEEE Computer Society standards projects is available from the IEEE Computer Society, 1730 Massachusetts Avenue NW, Washington, DC 20036-1903; Telephone: +1 202 371-0101; FAX: +1 202 728-9614. Working drafts of POSIX standards under development are also available from this office.

P1003.2 was prepared by the 1003.2 working group, sponsored by the Technical Committee on Operating Systems and Application Environments of the IEEE Computer Society. At the time this standard was approved, the membership of the 1003.2 working group was as follows:

**Technical Committee on Operating Systems
and Application Environments (TCOS)**

Chair: Jehan-François Pâris

TCOS Standards Subcommittee

Chair: Jim Isaak
Vice Chairs: Ralph Barker
David Dodge
Robert Bismuth
Hal Jespersen
Lorraine Kevra
Treasurer: Quin Hahn
Secretary: Shane McCarron

1003.2 Working Group Officials

Chair: Hal Jespersen
Vice Chair: Donald W. Cragun
Editors: Hal Jespersen (1986, 1988-1991)
Maggie Lee (1987-1988)
Secretaries: Helene Armitage (1988-1990)
Dave Grindeland (1991)
Robert J. Makowski (1987-1988)

Technical Reviewers

Helene Armitage	Ken Faubel	Gary Miller
Keith Bostic	Greger Leijonhufvud	Marc Teitelbaum
John Caywood	Bob Lenk	Donn Terry
Donald Cragun	Mark Levine	Teoman Topcubasi
David Decot	Shane McCarron	David Willcox

Working Group

Helene Armitage	Quin Hahn	Jim Oldroyd
Brian Baird	Michael J. Hannah	Mark Parenti
John R. Barr	Marjorie E. Harris	John Peace
Philippe Bertrand	David F. Hinnant	Jon Penner
Robert Bismuth	Leon M. Holmes	Gerald Powell
Jim Blondeau	Ron Holt	John Quarterman
James C. Bohem	Randall Howard	Joe Ramus
Kathy Bohrer	Steven A. James	Mike Ressler
Keith Bostic	Steve Jennings	Grover Righter
Phyllis Eve Bregman	Hal Jespersen	Andrew K. Roach
Peter Brouwer	Ronald S. Karr	Marco P. Roodzant
F. Lee Brown, Jr.	Lorraine C. Kevra	Seth Rosenthal
Jonathan Brown	Martin Kirk	Maude Sawyer

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

146	James A. Capps	Brad Kline	Norman K. Scherer
147	Bill Carpenter	Hiromichi Kogure	Glen Seeds
148	Steve Carter	David Korn	Jim Selkaitis
149	John Caywood	Rick Kuhn	Karen Sheaffer
150	Bob Claeson	Mike Lambert	Del Shoemaker
151	Mark Colburn	Maggie Lee	James Soddy
152	Donald W. Cragun	Perry Lee	Daniel Steinberg
153	Dave Decot	Greger Leijonhufvud	Scott A. Sutter
154	Terence S. Dowling	Bob Lenk	Ravi Tavakley
155	Stephen Dum	Mark Levine	Marc Teitelbaum
156	Dominic Dunlop	Gary Lindgren	Donn Terry
157	Mike Edmonds	John Lomas	Jack Thompson
158	Ron Elliott	Craig Lund	Teoman Topcubasi
159	Richard W. Elwood	Rod MacDonald	Eugene Tsuno
160	Hirsaki Eto	Dan Magenheimer	Geraldine Vitovitch
161	Fran Fadden	Robert J. Makowski	Carl vonLoewenfeldt
162	Ken Faubel	Shane P. McCarron	Mike Wallace
163	Martin C. Fong	Jim McGinness	Alan Weaver
164	Terance Fong	John McGrory	Larry Wehr
165	Glenn Fowler	Stuart McKaig	Bruce Weiner
166	Gary A. Gaudet	Sunil Mehta	N. Ray Wilkes
167	Al Gettier	Bill Middlecamp	David Willcox
168	Timothy D. Gill	Gary W. Miller	Neil Winton
169	Gregory Goddard	Jim Moe	David Woodend
170	Loretta Goudie	Yasushi Nakahara	Morten With
171	Dave Grindeland	Martha Nalebuff	Ken Witte
172	John Lawrence Gregg	Sonya D. Neuffer	John Wu
173	Jerry Gross	Landon Noll	Peggy Younger
174	Douglas A. Gwyn	Robin T. O'Neill	Hilary Zaloom

175 The following persons were members of the 1003.2 Balloting Group that approved
176 the standard for submission to the IEEE Standards Board:

177	Derek Kaufman	<i>X/Open Institutional Representative</i>	
178	Shane McCarron	<i>UNIX International Institutional Representative</i>	
179	Peter Collinson	<i>USENIX Association Institutional Representative</i>	
180	Scott Anderson	Carol J. Harkness	Jim R. Oldroyd
181	Helene Armitage	Craig Harmer	Craig Partridge
182	David Athersych	Dale Harris	Rob Peglar
183	Geoff Baldwin	Myron Hecht	John C. Penney
184	Jerome E. Banasik	Morris J. Herbert	Rand S. Phares
185	Steven E. Barber	David F. Hinnant	P. J. Plauger
186	Robert M. Barned	Lee A. Hollaar	Gerald Powell
187	David R. Bernstein	Ronald Holt Jr.	Scott E. Preece
188	Kabekode V. S. Bhat	Randall Howard	James M. Purtilo
189	Robert Bismuth	Jim Isaak	J. S. Quarterman
190	Jim Blondran	Richard James	Wendy Rauch-Hindin
191	Robert Borochoff	Hal Jespersen	Brad Rhoades
192	Keith Bostic	Greg Jones	Christopher J. Riddick
193	James P. Bound	Michael J. Karels	Andrew K. Roach
194	Joseph Boykin	Lorraine C. Kevra	Arnold Robbins
195	Kevin Brady	Alan W. Kiecker	R. Hughes Rowlands
196	Phyllis Eve Bregman	Jeff Kimmel	Robert Sarr

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

197	A. Winsor Brown	M. J. Kirk	Norman Schneidewind
198	F. Lee Brown Jr.	Kenneth C. Klingman	Wolfgang Schwabl
199	Luis-Felipe Cabrera	Joshua W. Knight	Richard Scott
200	Nicholas A. Camillone	David Korn	Glen Seeds
201	Andres Caravallo	Takahiko Kuki	Dan Shia
202	Steven L. Carter	Robin B. Lake	Roger Shimada
203	John Caywood	Mike Lambert	Mukesh Singhal
204	Kilnam Chon	Doris Lebovits	Richard Sniderman
205	Chan F. Chong	Maggie Lee	Steven Sommars
206	Robert L. Claeson	Greger Leijonhufvud	Bryan W. Sparks
207	Mark Colburn	Robert M. Lenk	Richard Stallman
208	Kenneth N. Cole	David Lennert	Daniel Steinberg
209	Richard Cornelius	Mark E. Levine	Douglas H. Steves
210	William M. Corwin	Kevin Lewis	Peter Sugar
211	Mike R. Cossey	Kin F. Li	Scott A. Sutter
212	William Cox	James P. Lonjers	Ravi Tavakley
213	Donald W. Cragun	Joseph F. P. Luhukay	Donn Terry
214	Terence Dowling	Paul Lustgarten	Gary F. Tom
215	Stephen A. Dum	Ron Mabe	A. T. Twigger
216	John D. Earls	Robert J. Makowski	Mark-Rene Uchida
217	Ron Elliott	Roger J. Martin	L. David Umbaugh
218	Richard W. Elwood	Joberto S. B. Martins	Michael W. Vannier
219	David Emery	Yoshihiro Matsumoto	M. B. Wagner
220	Philip H. Enslow	Shane McCarron	John W. Walz
221	Ken Faubel	Martin J. McGowan III	Alan G. Weaver
222	Terence Fong	Marshall Kirk McKusick	Larry Wehr
223	Ed Frankenberry	Robert W. McWhirter	Bruce Weiner
224	John A. Gertwagen	Doug Michels	Brian Weis
225	Al Gettier	Gary W. Miller	Peter J. Weyman
226	Michel Gien	James M. Moe	Andrew E. Wheeler
227	Gregory W. Goddard	J. W. Moore	David Willcox
228	Robert C. Groman	Anita Mundkur	Jeff Wubik
229	Judy Guist	Martha Nalebuff	Oren Yuen
230	Gregory Guthrie	Fred Noz	Jason Zions
231	Michael J. Hannah	Alan F. Nugent	

232 When the IEEE Standards Board approved this standard on *<date to be pro-*
233 *vided>*, it had the following membership:

234 (to be pasted in by IEEE)

Information technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities

Section 1: General

1.1 Scope

This standard defines a standard source code level interface to command interpretation, or “shell,” services and common utility programs for application programs. These services and programs are complementary to those specified by ISO/IEC 9945-1: 1990 {8}, hereinafter referred to as “POSIX.1 {8}.”

The standard has been designed to be used by both application programmers and system implementors. However, it is intended to be a reference document and not a tutorial on the use of the services, the utilities, or the interrelationships between the utilities.

The emphasis of this standard is on the shell and utility functionality required by application programs (including “shell scripts”) and not on the direct interactive use of the shell command language or the utilities by humans.

Portions of this standard comprise optional language bindings to system service interfaces. See, for example, the C Language Bindings Option in Annex B. This standard is intended to describe language interfaces and utilities in sufficient detail so that an application developer can understand the required interfaces without access to the source code of existing implementations on which they may be based. Therefore, it does not attempt to describe the source programming language or internal design of the utilities; they should be considered “black boxes” that exhibit the described functionality.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

For language interfaces, or functions, this standard has been defined exclusively at the source code level. The objective is that a conforming portable application source program can be translated to execute on a conforming implementation. The standard assumes that the source program may need to be retranslated to produce target code for a new environment prior to execution in that environment.

There is no requirement that the base operating system supporting the shell and utilities be one that fully conforms to ISO/IEC 9945-1: 1990 {8}. (The base system could contain a subset of POSIX.1 {8} functionality, enough to support the requirements for this standard, as described in 2.9.1, but that could not claim full conformance to all of POSIX.1 {8}.) Furthermore, there is no requirement that the shell command interpreter or any of the standard utilities be written as POSIX.1 {8} conforming programs, or be written in any particular language.

Although not requiring a fully conforming POSIX.1 {8} base, this standard is based upon documentation and the knowledge of existing programs that assume an interface and architecture similar to that described by POSIX.1 {8}. Any questions regarding the definition of terms or the semantics of an underlying concept should be referred to POSIX.1 {8}.

1.1.1 Scope Rationale. *(This subclause is not a part of P1003.2)*

This standard is one of a family of related standards. The term POSIX is correctly used to describe this family, and not only its foundation, the operating system interfaces of POSIX.1 {8}. Therefore, POSIX.2 could colloquially be described as the “POSIX Shell and Tools Standard.”

The interfaces documented for this standard are to and from high-level language application programs and to and from the utilities themselves; the standard does not directly address the interface with users.

The “source code” interface to the command interpreter is defined in terms of high-level language functions in 7.1.1 or 7.1.2 (such as *system()*, B.3.1, or *popen()*, B.3.2). There are also other function interfaces, such as those for matching regular expressions in 7.3 (*regcomp()* in B.5). Many of the utilities in this standard, and the shell itself, also accept their own command languages or complex directives as input data, which is also referred to as source code. This data, an ordered series of characters, may be stored in files, or “scripts,” that are portable between systems without true recompilation. However, just as with POSIX.1 {8}, the standard addresses only the issue of source code portability between systems; applications using these calls may have to be recompiled or translated when moving from one system to another.

There has been considerable debate concerning the appropriate scope of the work represented by this standard. The following are rational alternatives that have been evaluated:

- (1) Define the shell and tools as extensions to POSIX.1 {8}. This would require a full conforming POSIX.1 {8} system as a base for the new facilities described here. Vocal proponents for this view have been the members of the POSIX.3 working group, who foresaw difficulties in

producing a verification suite standard without having a known operating system base.

- (2) Decouple the shell and tools entirely from POSIX.1 {8}. This would potentially allow the standard to be implemented on such popular operating systems as MVS/TSO, VM/CMS, MS/DOS, VMS, etc. Those systems would not have to provide every minor detail of the POSIX.1 {8} language interfaces to conform under this model—only enough to support the shell and tools.
- (3) Compromise between options 1 and 2. Base the standard on an interface *similar* to POSIX.1 {8}, but don't require full conformance. A simple example would be a Version 7 UNIX System, which could not conform to POSIX.1 {8} without considerable modification. However, a vendor could support all of the features of this standard without changing its kernel or binary compatibility. Another example would be a system that conformed to all stated POSIX.1 {8} interfaces, but that didn't have a fully conforming C Standard {7} compiler. The difficulty with this option is that it makes the stated goal of the working group a bit fuzzier and increases the amount of analysis required for the features included.

The working group selected option 3 as its goal. It chose to retain the full UNIX system-like orientation, but did not wish to arbitrarily deprive legitimate systems that could *almost* conform. No useful feature of shells or commonly-used utilities were discarded to accommodate nonconforming base systems; on the other hand, no deliberate obstacles were arbitrarily erected. Furthermore, POSIX.1 {8} is still required for its definitions and architectural concepts, which are purposely not repeated in this standard.

One concrete example of how the two standards interrelate is in the usage of POSIX.1 {8} function names in the descriptions of utilities in POSIX.2. There are a number of historical commands that directly mapped into one of the UNIX system calls. For example: `chmod` and `chmod()`; `ln` and `link()`. The POSIX.2 working group was faced with the problem of having to define all of the complex interactions “behind the scenes” for some simple commands. Creating a file, for example, involves many POSIX.1 {8} concepts, including processes, user IDs, multiple group permissions (which are optional), error conditions, etc. Rather than enumerating all of these interactions in many places, the POSIX.2 group chose to employ the POSIX.1 {8} function descriptions, where appropriate. See the `chmod` utility in 4.7 as an example. The utility description includes the phrase:

... performing actions equivalent to the `chmod()` function as defined in the POSIX.1 {8} `chmod()` function:

This means that the POSIX.2 implementor has to read the POSIX.1 {8} `chmod()` description and fully understand all of its functionality, requirements, and side effects, which now don't have to be repeated here. (Admittedly, this makes the POSIX.2 standard a bit more difficult to read, but the working group felt that precision transcended the need for readable or semi-tutorial documents.)

The Introduction states that one of the goals of the working group was: “This interface should be implementable on conforming POSIX.1 {8} systems.” This

implies that the working group has attempted to ensure that no additional functionality or extension is required to implement this standard on the base defined by POSIX.1 {8}. This is not to say that extensions are not allowed, but that they should not be necessary. The goal “(7) Utilities and standards for the installation of applications” was once interpreted to mean that an elaborate series of tools was required to install and remove applications, based on complex description files and system databases of capabilities. An attempt to provide this was rejected by the balloting group and that type of system is now being evaluated by the POSIX.7 System Administration group. However, the original goal remains in the list, because many of the standard utilities are, in fact, targeted specifically for application installation—make, c89, lex, etc.

1.1.1.1 Existing Practice. *(This subclause is not a part of P1003.2)*

The working group would have been very happy to develop a standard that allowed all historical implementations (i.e., those existing prior to the time of publication) to be fully conforming and all historical applications to be Strictly Conforming POSIX Shell Applications without requiring any changes. Some modifications will be required to reconcile the specific differences between historical implementations; there are many divergent versions of UNIX systems extant and applications have sometimes been written to take advantage of features (or bugs) on specific systems. Therefore, the working group established a set of goals to maximize the value of the standard it eventually produced. These goals are enumerated in the following subclauses. They are listed in approximate priority sequence, where the first subclause is the most important portability goal.

1.1.1.1.1 Preserve Historical Applications

The most important priority was to ensure that historical applications continued to operate on conforming implementations. This required the selection of many utilities and features from the most prevalent historical implementations. The working group is relying on the following factors:

- (1) Many inconsistent historical features will still be supported as *obsolescent*.
- (2) Common features of System V and BSD will continue to be supported by their sponsors, even if they aren’t included here (just as long as they are not prevented from existing).

Therefore, the standard was written so that the large majority of well-written historical applications should continue to operate as Conforming POSIX Shell Applications Using Extensions.

1.1.1.1.2 Clean Up the Interfaces

The working group chose to extend the benefits of historical UNIX systems by making limited improvements to the utility interfaces; numerous complaints have been heard over the years about the inconsistencies in the command line interface, which have allegedly made it harder for novice users. Given the constraints

of **Preserve Historical Applications**, the working group has made the following general modifications:

- (1) Utilities have been extended to deal with differences in character sets, collating sequences, and some cultural aspects relating to the locale of the user. (Examples: new features in regular expressions; new formatting options in `date`; see 4.15.)
- (2) The utility syntax guidelines in 2.10.2 have been applied to almost all of the utilities to promote a consistent interface. The guidelines themselves have been loosened up a bit from their counterparts in the *SVID*. In many cases historical utilities have not conformed with these guidelines (which were written considerably later than the utilities themselves). The older interfaces have been maintained in the standard as obsolescent features. (Examples: `join`, `sort`.) However, in some cases, such as `dd` and `find`, such major surgery was required that the working group decided to leave the historical interfaces as is. “Fixing” the interface would mean replacing the command, which would not help applications portability. So, fixing was limited to relatively minor abuses of the new guidelines, where reasonable consistency could be achieved while still maintaining the general type of interface of the historical version.
- (3) Features that were not generally portable across machine architectures or systems have been removed or marked obsolescent and new, more portable interfaces have been introduced. (Examples: the octal number methods of describing file modes in `chmod` and other utilities have been marked obsolescent; the symbolic “`ugo`” method has been extended to other utilities, such as `umask`.)
- (4) Features that have proved to be popular in some specific UNIX system variants have been adopted. (Examples: `diff -c`, which originated in BSD systems, and the “new” `awk`, from System V.) Such features were selected given the requirements for balloting group consensus; the features had to be used widely enough to balance accusations of “creeping featurism” and violations of the UNIX system “tools philosophy.”
- (5) Unreasonable inconsistencies between otherwise similar interfaces have been reconciled. (Example: methods of specifying the patterns to the three `grep`-related utilities have been made more consistent in the standard’s single `grep`.)
- (6) When irreconcilable differences arose between versions of historical utilities, new interfaces (utility names or syntax) were sometimes added in their places. The working group resisted the urge to deviate significantly from historical practice; the new interfaces are generally consistent with the philosophy of historical systems and represent comparable functionality to the interfaces being replaced. In some cases, System V and BSD had diverged (such as with `echo` and `sum`) so significantly that no compromises for a common interface were possible. In these cases, either the divergent features were omitted or an entirely new command name was selected (such as with `printf` and `cksum`).

- (7) Arbitrary limits to utility operations have been removed. (Example: some historical `ed` utilities have very limited capabilities for dealing with large files or long input lines.)
- (8) Arbitrary limitations on historical extensions have been eliminated. (Example: regular expressions have been described so that the popular `\< ... \>` extension is allowed.)
- (9) Input and output formats have been specified in more detail than historical implementations have required, allowing applications to more effectively operate in pipelines with these utilities. (Example: `comm`.)

Thus, in many cases the working group could be accused of “violating Existing Practice,” and in fact received some balloting objections to that effect from implementors (although rarely from users or application developers). The working group was sensitive to charges that it was engaged in arbitrary software engineering rather than merely codifying existing practice. When changes were made, they were always written to preserve historical applications, but to move new conforming applications into a more consistent, portable environment. This strategy obviously requires changes to historical implementations; the working group carefully evaluated each change, weighing the value to users against the one-time costs of adding the new interfaces (and of possibly breaking applications that took advantage of bugs), generally siding with the users when the costs to implementations and applications was not excessively high.

In some cases, changes were reluctantly made that could conceivably break some historical applications; the working group allowed these only in the face of practices it considered rare or significantly misguided.

1.1.1.1.3 Allow Historical Conforming Applications

It is likely that many historical shell scripts will be Strictly Conforming POSIX.2 Applications without requiring modifications. Developers have long been aware of the differences among the historical UNIX system variants and have avoided the nonportable aspects to increase the scope of their applications’ marketplace. However, the previous goal of a consistent interface was considered to be quite important, so there will be modifications required to some applications if they wish to be maximally portable in the future.

1.1.1.1.4 Preserve Historical Implementations

As explained in 1.1.1.1.2, the requirements for portability and a consistent interface have caused the working group to add new utilities and features. No historical implementations contained all of the attributes required by the working group. Therefore, this lowest priority goal fell victim to the preceding goals, and every known historical implementation will require some modifications to conform to this standard.

The working group took care to ensure that the implementations could add the new or modified features without breaking the operation of existing applications. (Note that the standard utilities are not considered applications in this regard,

but are part of the implementation. In fact, many or most of the utilities named by this standard will have to change to some extent.)

1.1.1.2 Outside the Scope. *(This subclause is not a part of P1003.2)*

The following areas are outside the scope of this standard. This subclause explains more of the rationale behind the exclusions. (It should be noted that this is not an official list. It was not part of the Project Authorization Request submitted to the IEEE, but was devised as a guide to keep the working group discussions on track.)

- (1) *Operating system administrative commands (privileged processes, system processes, daemons, etc.).*

The working group followed the lead of the POSIX.1 {8} group in this instance. Administrative commands were felt to be too implementation dependent and not useful for application portability. Subsequent to this decision, a separate POSIX.7 working group was formed to deal with this area of “operator portability.” It is anticipated that utilities needed for system administration will be closely coordinated with the POSIX.2 working group.

- (2) *Commands required for the installation, configuration, or maintenance of operating systems or file systems.*

This area is similar to item (1). System installation is contrasted against the application installation portion of the Scope by its orientation to installing the operating system itself, versus application programs. The exclusion of operating system installation facilities should not be interpreted to mean that the application installation procedures *cannot* be used for installing operating system components. The proposed interface for this area encountered stiff resistance from the balloting group in Draft 8 and was temporarily withdrawn. As described in Annex E.4, a decision of the balloting group is pending on whether to begin work on a supplement to this standard (POSIX.2b) for application installation.

- (3) *Networking commands.*

These were excluded because they are deeply involved with other standards making bodies and are probably too complicated. In this case, several working groups were formed within the POSIX family to deal with this. It is anticipated that utilities needed for networking, if any, will be closely coordinated with the POSIX.2 working group. (In early drafts of this standard, which predated the formation of the networking-specific POSIX working groups, the historical “UNIX system to UNIX system copy [UUCP]” programs and protocols were included. These descriptions have been removed in deference to a more appropriate working group.)

- (4) *Terminal control or user-interface programs (e.g., visual shells, visual editors, window managers, command history mechanisms, etc.).*

This is probably the most contentious exclusion. A common complaint about many UNIX systems is how they're not very "user friendly." Some people have hoped that the interface to users could be standardized with mice, icon-based desktop metaphors, and so forth. This standard neatly sidesteps those concerns by reminding its audience that it is an application portability standard, and therefore has little relationship to the manner in which users manage their terminals.

However, this guideline was not meant to apply to applications. It is perfectly reasonable for an application to assume it can have a user interacting with it. That is why such facilities as displaying strings (with `printf`) without `<newline>`s, `stty`, and various prompting utilities are included in the standard.

The interfaces in this standard are very oriented to command lines being issued by shell scripts, or through the `system()` or `popen()` functions. Therefore, interactive text editors, pagers, and other user interface tools have been omitted for now. Alternatively, other standards bodies, such as X3H3.6 and the IEEE TCOS P1201 working group, are devising interfaces that could possibly be more useful and long-lived than any prescribed by POSIX.2.

There is one area of this subject that will be addressed by POSIX.2. The scope of the working group has been expanded to include what is being termed the *User Portability Extension*, POSIX.2a. This will be published as a supplement to this standard and have the goal of providing a portable environment for relatively expert time-sharing or software development users. It will not attempt to deal with mice or windows or other advanced interfaces at this time, but should cover many of the terminal-oriented utilities, such as a full-screen editor, currently avoided by this edition of POSIX.2.

(5) *Graphics programs or interfaces.*

See the comments on user interface, above.

(6) *Text formatting programs or languages.*

The existing text formatting languages are generally too primitive in scope to satisfy many users, who have relied on a myriad of macro languages. There is an ISO standard text description language, SGML, but this has had insufficient exposure to the UNIX system community for standardization as part of POSIX at this time.

(7) *Database programs or interfaces (e.g. SQL, etc.).*

These interfaces are the province of other standards bodies.

1.1.1.3 Language-Independent Descriptions. *(This subclause is not a part of P1003.2)*

The POSIX.1 {8} and POSIX.5 working groups are currently engaged in developing the model for language-independent descriptions of system services. When complete, it will allow the C language bias of the POSIX.1 {8} standard to be excised and C will take its place among other language bindings that interface with the core services descriptions. The POSIX.2 working group did not wish to duplicate effort, and has therefore waited until POSIX.1 {8} achieves progress in this area. Thus, like the first version of POSIX.1 {8}, the initial drafts of POSIX.2 start life as a C-only standard, with language independence scheduled to be included in a later draft. Fortunately, this standard is substantially less involved with C than POSIX.1 {8} is. In fact, all of the C interfaces are entirely optional.

1.1.1.4 Base Documents. *(This subclause is not a part of P1003.2)*

The working group consulted a number of documents in the course of its deliberations, to select utilities and features. There were five primary documents that started off the process:

- (1) *The System V Interface Definition (SVID)*, Issue 2, Volume 2.
- (2) *The X/Open Portability Guide, (XPG)*, Issues II and III, Volume 1.
- (3) *The UNIX User's Reference Manual*, 4.3 Berkeley Software Distribution, Virtual VAX-11 Version. (The printed documentation as well as the online versions provided with the BSD "Tahoe" and "Reno" distributions were considered as one base document for the POSIX.2 work.)
- (4) *The KornShell Command and Programming Language*, by Bolsky and Korn.
- (5) *The AWK Programming Language*, by Aho, Kernighan, and Weinberger.

The XPG was used most heavily in initial deliberations about which utilities and features to include. The X/Open companies had done a very thorough job in analyzing the SVID and other standards to compile a list of the most useful and portable utilities. They carefully marked many features that had portability problems and the working group avoided them for this standard.

AT&T, X/Open, and Berkeley provided machine-readable documentation for the use of the working group. However, due to very substantial differences in formatting standards, there is little resemblance between some of the utilities described here and their cousins in the SVID, XPG, and BSD user manual. Nevertheless, early usage of these documents was an invaluable aid in the production of the standard and the POSIX.2 working group extends its sincere thanks to all three organizations for their generous cooperation.

The biggest divergence in POSIX.2's documentation has been its philosophy of fully specifying interfaces. The SVID and XPG are oriented solely towards application portability. Implementors would have a difficult time writing some of these utilities from the descriptions alone. In fact, both documents freely rely on the

potential implementors licensing the source code for the reference systems to complete the specification. The POSIX.2 standard, on the other hand, also has implementors in its audience and it strove to expand its descriptions wherever useful and feasible. For example, it makes use of BNF grammars to describe complex syntaxes. It attempts to describe the interactions between options, operands, and environment variables, where conflicts can exist. It also attempts to describe all of the useful utility input and output formats. The goal here was to allow application developers to write filters or other programs that could parse the output of any of these utilities or to provide meaningful input from their programs. To the working group's knowledge, this is a task never before attempted for the historical UNIX system commands—the source code was always so readily available to anyone who really needed to know this information.

The two commercial books listed were used as reference materials in preparing information on the shell and the *awk* language that was more recent and complete than AT&T's or X/Open's documentation.

1.1.1.5 History. *(This subclause is not a part of P1003.2)*

The 1984 */usr/group Standard* was originally intended to include the shell and user level commands. However, the */usr/group* (now known as “UniForum”) Standards Committee was unable to begin this effort, due to the complexity of the system call and library functions that it eventually did publish.

A shell was referred to in the *system()* function defined by *ANSI/X3.159-1989 Programming Language C Standard*, but no syntax for the shell command language was attempted.

As the first version of POSIX.1 {8} neared completion, it became apparent that the usefulness of POSIX would be diminished if no shell or utilities were defined. Therefore, the POSIX.2 working group was formed in January 1986 at the Denver, Colorado, meeting of POSIX.1 {8} to address this concern.

The progress of the working group has seemed rather slow during the more than three years of its existence. This is primarily because its membership had substantial overlap with the POSIX.1 {8} working group; for example, the Chair of POSIX.2 was also the Technical Editor of POSIX.1 {8} (and POSIX.2 as well!) at the time. And, meetings were arbitrarily shortened to allow the POSIX.1 {8} group to move forward as quickly as possible.

1.1.1.6 Internationalization. *(This subclause is not a part of P1003.2)*

Some of the utilities and concepts described in this standard contain requirements that standardize multilingual and multicultural support. Most of the internationalized support for this standard was proposed by the UniForum Technical Committee Subcommittee on Internationalization, at the request of the POSIX.2 working group.

UniForum, a nonprofit organization, organizes subcommittees of Technical Committees to do standards research on different topics pertinent to POSIX. The

UniForum Subcommittee on Internationalization is one such group. It was formed to propose and promote standard internationalized extensions to POSIX-based systems. The POSIX.2 working group and the UniForum Subcommittee on Internationalization coordinated their work by the use of liaison members, who attended the meetings of both groups. The interaction between the two groups started when POSIX.2 asked the Subcommittee on Internationalization to provide internationalized support for regular expressions. Later, the Subcommittee on Internationalization was charged with identifying areas in the standard needing changes for internationalized support and proposing those changes.

1.1.1.7 Test Methods. *(This subclause is not a part of P1003.2)*

The POSIX.3 working group has worked on a test methods specification for verifying conformance to POSIX standards in general and POSIX.1 {8} and POSIX.2 in particular. Test methods for POSIX.2 should be published as a separate document¹⁾ sometime after POSIX.2 is approved.

1.1.1.8 Organization of the Standard. *(This subclause is not a part of P1003.2)*

The standard document is organized into sections. Some of these, such as the Scope in 1.1, are mandated by ISO/IEC, the IEEE, and other standards bodies. The remainder of the document is organized into small sections for the convenience of the working group and others. It has been suggested that all of the utility descriptions (and maybe the functions, too) should be lumped into one large section, all in alphabetical order. This would presumably make it easier for some users to use the document as a reference document. The working group deliberately chose to not organize it in this way, for the following reasons:

- (1) Certain sections are optional. It is more convenient for the document's internal references, and also for people specifying systems, if these optional sections are in large pieces, rather than a detailed list of utility names.
- (2) Future supplements to this standard will be adding new utilities that will also be optional. It would be confusing to try to merge documents at a level below major sections (chapters).

1) See the Foreword for information on the activities of other POSIX working groups.

1.2 Normative References

The following standards contain provisions which, through references in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of this International Standard 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.

- {1} ISO/IEC 646: 1983,²⁾ *Information processing—ISO 7-bit coded character set for information interchange.*
- {2} ISO 1539: 1980, *Programming languages—FORTRAN.*
- {3} ISO 4217: 1987, *Codes for the representation of currencies and funds.*
- {4} ISO 4873: 1986, *Information processing—ISO 8-bit code for information interchange—Structure and rule for implementation.*
- {5} ISO 8859-1: 1987, *Information processing—8-bit single-byte coded graphic character sets—Part 1: Latin alphabet No. 1.*
- {6} ISO 8859-2: 1987, *Information processing—8-bit single-byte coded graphic character sets—Part 2: Latin alphabet No. 2.*
- {7} ISO/IEC 9899: 1990, *Information processing systems—Programming languages—C.*
- {8} ISO/IEC 9945-1: 1990, *Information technology—Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language]*

1.3 Conformance

1.3.1 Implementation Conformance

1.3.1.1 Requirements

A conforming implementation shall meet all of the following criteria:

²⁾ Under revision. (This notation is meant to explicitly reference the 1990 Draft International Standard version of ISO/IEC 646.)

ISO/IEC documents can be obtained from the ISO office, 1, rue de Varembe, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse.

- (1) The system shall support all required interfaces defined within this standard. These interfaces shall support the functional behavior described herein. The system shall provide the shell command language described in Section 3 and the utilities in Section 4.
- (2) The system may provide one or more of the following: the Software Development Utilities Option, the C Language Bindings Option, the C Language Development Utilities Option, the FORTRAN Development Utilities Option, or the FORTRAN Runtime Utilities Option. When an implementation claims that an optional facility is provided, all of its constituent parts shall be provided.
- (3) The system may provide additional or enhanced utilities, functions, or facilities not required by this standard. Nonstandard extensions should be identified as such in the system documentation. Nonstandard extensions, when used, may change the behavior of utilities, functions, or facilities defined by this standard. In such cases, the implementation's conformance document (see 2.2.1.2) shall define an execution environment (i.e., shall provide general operating instructions) in which an application can be run with the behavior specified by the standard. In no case shall such an environment require modification of a Strictly Conforming POSIX.2 Application.

1.3.1.2 Documentation

A conformance document with the following information shall be available for an implementation claiming conformance to this standard. The conformance document shall have the same structure as this standard, with the information presented in the appropriately numbered sections; sections 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 this standard, unless those extensions affect the behavior of a Strictly Conforming POSIX.2 Application; in such cases, the documentation required by the previous subclause shall be included.

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 software standards approved by ISO/IEC or any ISO/IEC member body that are available for use by a Conforming POSIX.2 Application. It should indicate whether it is based on a fully-conformant POSIX.1 {8} system. 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 symbolic values found in 2.13.2, stating values, the conditions under which those values can 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 this standard. This requirement shall be met by listing these features and providing either a specific reference to

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

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 customizable on each instantiation of the system, the implementation provider shall document the nature and permissible ranges of this variation. When information required by this standard is related to the underlying operating system and is already available in the POSIX.1 {8} conformance document, the implementation need not duplicate this information in the POSIX.2 conformance document, but may provide a cross-reference for this purpose.

The conformance document may specify the behavior of the implementation for those features where this standard states that implementations may vary or where features are identified as undefined or unspecified.

No specifications other than those described in this subclause (1.3.1.2) shall be present in the conformance document.

The phrase “shall be documented” in this standard means that documentation of the feature shall appear in the conformance document, as described previously, unless the system documentation is explicitly mentioned.

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

1.3.1.3 Conforming Implementation Options

The following symbolic constants, described in 2.13.2 reflect implementation options for this standard that could warrant requirement by Conforming POSIX.2 Applications, or in specifications of conforming systems, or both:

{POSIX2_SW_DEV}	The system supports the Software Development Utilities Option in Section 6.
{POSIX2_C_BIND}	The system supports the C Language Bindings Option in Annex B.
{POSIX2_C_DEV}	The system supports the C Language Development Utilities Option in Annex A.
{POSIX2_FORT_DEV}	The system supports the FORTRAN Development Utilities Option in Annex C.
{POSIX2_FORT_RUN}	The system supports the FORTRAN Runtime Utilities Option in Annex C.
{POSIX2_LOCALEDEF}	The system supports the creation of locales as described in 4.35.

Additional language bindings and development utility options may be provided in other related standards or in future revisions to this standard. In the former case, additional symbolic constants of the same general form as shown in this subclause should be defined by the related standard document and made available to the application, without requiring this POSIX.2 document to be updated.

1.3.2 Application Conformance

All applications claiming conformance to this standard fall within one of the following categories:

1.3.2.1 Strictly Conforming POSIX.2 Application

A Strictly Conforming POSIX.2 Application is an application that requires only the facilities described in this standard (including any required facilities of the underlying operating system; see 2.9.1). Such an application:

- (1) shall accept any implementation behavior that results from actions it takes in areas described in this standard as *implementation-defined* or *unspecified*, or where the standard indicates that implementations may vary;
- (2) shall not perform any actions that are described as producing *undefined* results;
- (3) for symbolic constants, shall accept any value in the range permitted by this standard, but shall not rely on any value in the range being greater than the minimums listed in this standard;
- (4) shall not use facilities designated as *obsolescent*;
- (5) is required to tolerate, and is permitted to adapt to, the presence or absence of optional facilities whose availability is indicated by the constants in 2.13.1, or that are described using the verb *may*. However, an application requiring a high-level language binding option can only be considered at best a Conforming POSIX.2 Application; see 1.3.2.2.

Within this standard, any restrictions placed upon a Conforming POSIX.2 Application shall also restrict a Strictly Conforming POSIX.2 Application.

1.3.2.2 Conforming POSIX.2 Application

The term Conforming POSIX.2 Application is used to describe either of the two following application types.

1.3.2.2.1 ISO/IEC Conforming POSIX.2 Application

An ISO/IEC Conforming POSIX.2 Application is an application that uses only the facilities described in this standard (including the implied facilities of the underlying operating system; see 2.9.1) and approved conforming language bindings for any ISO/IEC standard. Such an application shall include a statement of conformance that documents all options and limit dependencies, and all other ISO/IEC standards used.

1.3.2.2.2 <National Body> Conforming POSIX.2 Application

A <National Body> Conforming POSIX.2 Application differs from an ISO/IEC Conforming POSIX.2 Application in that it also may use specific standards of a single ISO/IEC member body referred to here as “<National Body>.” Such an application shall include a statement of conformance that documents all options and limit dependencies, and all other <National Body> standards used.

1.3.2.3 Conforming POSIX.2 Application Using Extensions

A Conforming POSIX.2 Application Using Extensions is an application that differs from a Conforming POSIX.2 Application only in that it uses nonstandard facilities that are consistent with this standard. Such an application shall fully document its requirements for these extended facilities, in addition to the documentation required of a Conforming POSIX.2 Application. A Conforming POSIX.2 Application Using Extensions shall be either an ISO/IEC Conforming POSIX.2 Application Using Extensions or a <National Body> Conforming POSIX.2 Application Using Extensions (see 1.3.2.2.1 and 1.3.2.2.2).

1.3.3 Conformance Rationale. *(This subclause is not a part of P1003.2)*

These conformance definitions are closely related to those in POSIX.1 {8}.

The terms *Conforming POSIX.2 Application* and its variants were selected to parallel the terms used in POSIX.1 {8}.

The descriptions of the ISO/IEC and <National Body> Conforming POSIX.2 Applications are similar to the same descriptions in POSIX.1 {8}. This is not a duplication of effort, as this standard relies on only a portion of POSIX.1 {8}, as explained in 1.1 and 2.9.1. Therefore conformance to POSIX.2 has to be described separately from any conformance options or requirements in POSIX.1 {8}.

A reference to a Language-Independent System Services Option was removed from the list of optional features that may be provided by the conforming implementation. There is no conformance value provided by that section, except as a reference point for functions actually provided by a real language binding. Therefore, the language binding sections are the ones that remain in the optional list. The Draft 8 section Language-Dependent Services for the C Programming Language was removed, as this subject is adequately, and appropriately, covered in Annex A.

The documentation requirement for implementation extensions (“shall define an execution environment”) is simply meant to require that system-wide or per-user configuration options or environment variables that affect the operation of applications that use the standard utilities and functions be described in the conformance document. For example, if setting the (imaginary) **LC_TRUTH** variable causes changes in the exit status of **true**, the conformance document must describe this condition and how to avoid it—say, by unsetting the variable in the login script.

616

For further rationale on the types of conformance, see the POSIX.1 {8} Rationale.

Section 2: Terminology and General Requirements

2.1 Conventions

2.1.1 Editorial Conventions

This standard uses the following editorial and typographical conventions. A summary of typographical conventions is shown in Table 2-1.

Table 2-1 – Typographical Conventions

Reference	Example
C-Language Data Type	<i>long</i>
C-Language Function	<i>system()</i>
C-Language Function Argument	<i>arg1</i>
C-Language Global External	<i>errno</i>
C-Language Header	<code><sys/stat.h></code>
C-Language Keyword	<code>#define</code>
Cross Reference: Annex	Annex A
Cross Reference: Clause	2.3
Cross Reference: Other Standard	ISO 9999-1 { <i>n</i> }
Cross Reference: Section	Section 2
Cross Reference: Subclause	2.3.4, 2.3.4.5, 2.3.4.5.6
Defined Term	(see text)
Environment Variable	PATH
Error Number	[EINTR]
Example Input	echo foo
Example Output	foo
Figure Reference	Figure 7
File Name	/tmp
Parameter	<i><directory pathname></i>
Special Character	<i><newline></i>
Symbolic Constant, Limit	{_POSIX_VDISABLE}, {LINE_MAX}
Table Reference	Table 6
Utility Name	awk
Utility Operand	<i>file_name</i>
Utility Option	-c
Utility Option with Option-Argument	-w <i>width</i>

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

The **Bold Courier** font is used to show brackets that denote optional arguments in a utility synopsis, as in

```
cut [-c list] [file_name]
```

These brackets shall not be used by the application unless they are specifically mentioned as literal input characters by the utility description.

There are two types of symbols enclosed in angle brackets (< >):

C-Language Headers The header name is in the Courier font, such as <sys/stat.h>. When coding C programs, the brackets are used as required by the language.

Parameters Parameters, also called *metavariables*, are in italics, such as <directory pathname>. The entire symbol, including the brackets, is meant to be replaced by the value of the symbol described within the brackets.

Numbers within braces, such as “POSIX.1 {8},” represent cross references to the Normative References clause (see 1.2). If the number is preceded by a B, it represents a Bibliographic entry (see Annex D). Bibliographic entries are for information only.

In some examples, the **Bold Courier** font is used to indicate the system’s output that resulted from some user input, shown in Courier.

Defined terms are shown in three styles, depending on context:

- (1) Terms defined in 2.2.1, 2.2.2, and 3.1 are expressed as subclause titles. Alternative forms of the terms appear in [brackets].
- (2) The initial appearances of other terms, applying to a limited portion of the text, are in *italics*.
- (3) Subsequent appearances of the term are in the Roman font.

Symbolic constants are shown in two styles: those within curly braces are intended to call the reader’s attention to values in <limits.h> and <unistd.h>; those without braces are usually defined by one or a few related functions. There is no semantic difference between these two forms of presentation.

Filenames and pathnames are shown in Courier. When a pathname is shown starting with “\$HOME/”, this indicates the remaining components of the pathname are to be related to the directory named by the user’s **HOME** environment variable.

The style selected for some of the special characters, such as <newline>, matches the form of the input given to the `localedef` utility (see 2.5.2). Generally, the characters selected for this special treatment are those that are not visually distinct, such as the control characters <tab> or <newline>.

Literal characters and strings used as input or output are shown in various ways, depending on context:

%, begin When no confusion would result, the character or string is rendered in the Courier font and used directly in the text.

'c' In some cases a character is enclosed in single-quote characters, similar to a C-language character constant. Unless otherwise noted, the quotes shall not be used as input or output.

"string" In some cases, a string is enclosed in double-quote characters, similar to a C-language string constant. Unless otherwise noted, the quotes shall not be used as input or output.

Defined names that are usually in lowercase, particularly function names, are never used at the beginning of a sentence or anywhere else that regular English usage would require them to be capitalized.

Parenthetical expressions within normative text also contain normative information. The general typographic hierarchy of parenthetical expressions is:

{ [()] }

The square brackets are most frequently used to enclose a parenthetical expression that contains a function name [such as *waitpid()*], with its built-in parentheses.

In some cases, tabular information is presented inline; in others it is presented in a separately-labeled Table. This arrangement was employed purely for ease of reference and there is no normative difference between these two cases.

Annexes marked as *normative* are parts of the standard that pose requirements, exactly the same as the numbered Sections, but have been moved to near the end of the document for clarity of exposition. *Informative* Annexes are for information only and pose no requirements. All material preceding page 1 of the document (the “front matter”) and the two indexes at the end are also only informative.

NOTES that appear in a smaller point size and are indented have one of two different meanings, depending on their location:

— When they are within the normal text of the document, they are the same as footnotes—informative, posing no requirements on implementations or applications.

— When they are attached to Tables or Figures, they are normative, posing requirements.

Text marked as examples (including the use of “e.g.”) is for information only. The exception to this comes in the C-language programs and program fragments used to represent algorithms, as described in 2.1.3.

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.

2.1.2 Grammar Conventions

Portions of this standard are expressed in terms of a special grammar notation. It is used to portray the complex syntax of certain program input. The grammar is based on the syntax used by the `yacc` utility (see A.3). However, it does not represent fully functional `yacc` input, suitable for program use: the lexical processing and all semantic requirements are described only in textual form. The grammar is not based on source used in any traditional implementation and has not been tested with the semantic code that would normally be required to accompany it. Furthermore, there is no implication that the partial `yacc` code presented represents the most efficient, or only, means of supporting the complex syntax within the utility. Implementations may use other programming languages or algorithms, as long as the syntax supported is the same as that represented by the grammar.

The following typographical conventions are used in the grammar; they have no significance except to aid in reading.

- The identifiers for the reserved words of the language are shown with a leading capital letter. (These are terminals in the grammar. Examples: `While`, `Case`.)
- The identifiers for terminals in the grammar are all named with uppercase letters and underscores. Examples: `NEWLINE`, `ASSIGN_OP`, `NAME`.
- The identifiers for nonterminals are all lowercase.

2.1.3 Miscellaneous Conventions

This standard frequently uses the C language to express algorithms in terms of programs or program fragments. The following shall be considered in reading this code:

- The programs use the syntax and semantics described by the C Standard [7].
- The programs are merely examples and do not represent the most efficient, or only, means of coding the interface. Implementations may use other programming languages or algorithms, as long as the results are the same as those achieved by the programs in this standard.
- C-language comments are informative and pose no requirements.

Further conventions are presented in:

- Utility Conventions, 2.10, describing utility and application command-line syntax
- File Format Notation, 2.12, describing the notation used to represent utility input and output

2.1.4 Conventions Rationale. *(This subclause is not a part of P1003.2)*

The C language was chosen for many examples because:

- It eliminates any requirement to document a different pseudocode.
- It is a familiar language to many of the potential readers of POSIX.2.
- It is the language most widely used for historical implementations of the utilities.

2.2 Definitions

2.2.1 Terminology

For the purposes of this standard, the following definitions apply:

2.2.1.1 can: The word *can* is to be interpreted as describing a permissible optional feature or behavior available to the application; the implementation shall support such features or behaviors as mandatory requirements.

2.2.1.2 conformance document: A document provided by an implementor that contains implementation details as described in 1.3.1.2.

2.2.1.3 implementation: An object providing to applications and users the services defined by this standard. The word *implementation* is to be interpreted to mean that object, after it has been modified in accordance with the manufacturer's instructions to:

- configure it for conformance with this standard;
- select some of the various optional facilities described by this standard, through customization by local system administrators or operators.

An exception to this meaning occurs when discussing conformance documentation or using the term *implementation defined*. See 2.2.1.4 and 1.3.1.2.

2.2.1.4 implementation defined: When a value or behavior is described by this standard as *implementation defined*, the implementation provider shall document the requirements for correct program construction and correct data in the use of that value or behavior. When the value or behavior in the implementation is designed to be variable or customizable on each instantiation of the system, the implementation provider shall document the nature and permissible ranges of this variation. (See 1.3.1.2.)

2.2.1.5 may: The word *may* is to be interpreted as describing an optional feature or behavior of the implementation that is not required by this standard, but there is no prohibition against providing it. A Strictly Conforming POSIX.2 Application 1

is permitted to use such features, but shall not rely on the implementation's actions in such cases. To avoid ambiguity, the reverse sense of *may* is not expressed as *may not*, but as *need not*.

2.2.1.6 obsolescent: Certain features are *obsolescent*, which means that they may be considered for withdrawal in future revisions of this standard. They are retained in this version because of their widespread use. Their use in new applications is discouraged.

2.2.1.7 shall: In this standard, the word *shall* is to be interpreted as a requirement on the implementation or on Strictly Conforming POSIX.2 Applications, where appropriate.

2.2.1.8 should: With respect to implementations, the word *should* is to be interpreted as an implementation recommendation, but not a requirement. With respect to applications, the word *should* is to be interpreted as recommended programming practice for applications and a requirement for Strictly Conforming POSIX.2 Applications.

2.2.1.9 system documentation: All documentation provided with an implementation, except the conformance document. Electronically distributed documents for an implementation are considered part of the system documentation.

2.2.1.10 undefined: A value or behavior is *undefined* if the standard imposes no portability requirements on applications for erroneous program construction, erroneous data, or use of an indeterminate value. Implementations (or other standards) may specify the result of using that value or causing that behavior. An application using such behaviors is using extensions, as defined in 1.3.2.3.

2.2.1.11 unspecified: A value or behavior is *unspecified* if the standard imposes no portability requirements on applications for a correct program construction or correct data. Implementations (or other standards) may specify the result of using that value or causing that behavior. An application requiring a specific behavior, rather than tolerating any behavior when using that functionality, is using extensions, as defined in 1.3.2.3.

2.2.1.12 Terminology Rationale (*This subclause is not a part of P1003.2*)

Most of these terms were adapted from their POSIX.1 {8} counterparts with little modification.

The reader is referred to the definition of *program* in 2.2.2.119 to understand the expression “program construction.” The use of *program* in this standard is differentiated from POSIX.1 {8}'s emphasis only on high level languages by this standard's broader concern with utility and command language interactions. Included in the scope of program construction are:

- (1) Shell command language
- (2) Command arguments
- (3) Regular expressions, of various types
- (4) Command input language syntax, such as `awk`, `bc`, `ed`, `lex`, `make`, `sed`, and `yacc`. Some of these are so complex that they rival traditional high level languages.

The usage of *can* and *may* were selected to contrast optional application behavior (can) against optional implementation behavior (may).

The term *supported* was removed from Draft 8; it had originally been copied from the POSIX.1 {8} document, but it later became clear that its requirement for function “stubs” for unsupported functions made little sense in this standard. The term *support* therefore reverts to its English-language meaning.

The term *obsolescent* was changed to *deprecated* in some earlier drafts, but it was restored to match POSIX.1 {8}’s use of the term. It means “do not use this feature in new applications.” The obsolescence concept is not an ideal solution, but was used as a method of increasing consensus: many more objections would be heard from the user community if some of these historical features were suddenly withdrawn without the grace period obsolescence implies. The phrase “may be considered for withdrawal in future revisions” implies that the result of that consideration might in fact keep those features indefinitely if the predominance of applications does not migrate away from them quickly.

2.2.2 General Terms

For the purposes of this standard, the following definitions apply.

2.2.2.1 absolute pathname: See *pathname resolution* in 2.2.2.104.

2.2.2.2 address space: The memory locations that can be referenced by a process. [POSIX.1 {8}]

2.2.2.3 affirmative response: An input string that matches one of the responses acceptable to the LC_MESSAGES category keyword `yesexpr`, matching an extended regular expression in the current locale; see 2.5.

2.2.2.4 <alert>: A character that in the output stream shall indicate that a terminal should alert its user via a visual or audible notification. 1

The <alert> shall be the character designated by ‘\a’ in the C language binding. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the alert function.

2.2.2.5 angle brackets: The characters “<” (*left-angle-bracket*) and “>” (*right-angle-bracket*).

When used in the phrase “enclosed in angle brackets” the symbol “<” shall immediately precede the object to be enclosed, and “>” shall immediately follow it. When describing these characters in 2.4, the names <less-than-sign> and <greater-than-sign> are used.

2.2.2.6 appropriate privileges: An implementation-defined means of associating privileges with a process with regard to the function calls and function call options defined in POSIX.1 {8} that need special privileges.

There may be zero or more such means. [POSIX.1 {8}]

2.2.2.7 argument: A parameter passed to a utility as the equivalent of a single string in the *argv* array created by one of the POSIX.1 {8} *exec* functions.

See 2.10.1 and 3.9.1.1. An argument is one of the options, option-arguments, or operands following the command name.

2.2.2.8 asterisk: The character “*”.

2.2.2.9 background process: A process that is a member of a background process group. [POSIX.1 {8}]

2.2.2.10 background process group: Any process group, other than a foreground process group, that is a member of a session that has established a connection with a controlling terminal. [POSIX.1 {8}]

2.2.2.11 backquote: The character “`”, also known as a *grave accent*.

2.2.2.12 backslash: The character “\”, also known as a *reverse solidus*.

2.2.2.13 <backspace>: A character that normally causes printing (or displaying) to occur one column position previous to the position about to be printed.

The <backspace> shall be the character designated by ‘\b’ in the C language binding. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the backspace function. The <backspace> character defined here is not necessarily the ERASE special character defined in POSIX.1 {8} 7.1.1.9.

2.2.2.14 basename: The final, or only, filename in a pathname.

2.2.2.15 basic regular expression: A pattern (sequence of characters or symbols) constructed according to the rules defined in 2.8.3.

2.2.2.16 <blank>: One of the characters that belong to the blank character class as defined via the LC_CTYPE category in the current locale.

In the POSIX Locale, a <blank> is either a <tab> or a <space>.

2.2.2.17 blank line: A line consisting solely of zero or more <blank>s terminated by a <newline>.

See also *empty line* (2.2.2.44).

2.2.2.18 block special file: A file that refers to a device.

A block special file is normally distinguished from a character special file by providing access to the device in a manner such that the hardware characteristics of the device are not visible. [POSIX.1 {8}]

2.2.2.19 braces: The characters “{” (*left brace*) and “}” (*right brace*), also known as *curly braces*.

When used in the phrase “enclosed in (curly) braces” the symbol “{” shall immediately precede the object to be enclosed, and “}” shall immediately follow it. When describing these characters in 2.4, the names <left-brace> and <right-brace> are used.

2.2.2.20 brackets: The characters “[” (*left-bracket*) and “]” (*right-bracket*), also known as *square brackets*.

When used in the phrase “enclosed in (square) brackets” the symbol “[” shall immediately precede the object to be enclosed, and “]” shall immediately follow it. When describing these characters in 2.4, the names `<left-square-bracket>` and `<right-square-bracket>` are used.

2.2.2.21 built-in utility: A utility implemented within a shell.

The utilities referred to as *special built-ins* have special qualities, described in 3.14. Unless qualified, the term *built-in* includes the special built-in utilities.

The utilities referred to as *regular built-ins* are those named in Table 2-2. As indicated in 2.3, there is no requirement that these utilities be actually built into the shell on the implementation, but that they do have special command-search qualities.

2.2.2.22 byte: An individually addressable unit of data storage that is equal to or larger than an octet, used to store a character or a portion of a character; see 2.2.2.24.

A byte is composed of a contiguous sequence of bits, the number of which is implementation defined. The least significant bit is called the *low-order* bit; the most significant is called the *high-order* bit. [POSIX.1 {8}]

NOTE: This definition of *byte* is actually from the C Standard {7} because POSIX.1 {8} merely references it without copying the text. It has been reworded slightly to clarify its intent without introducing the C Standard {7} terminology “basic execution character set,” which is inapplicable to this standard. It deviates intentionally from the usage of *byte* in some other standards, where it is used as a synonym for *octet* (always eight bits). On a POSIX.1 {8} system, a byte may be larger than eight bits so that it can be an integral portion of larger data objects that are not evenly divisible by eight bits (such as a 36-bit word that contains 4 9-bit bytes).

2.2.2.23 <carriage-return>: A character that in the output stream shall indicate that printing should start at the beginning of the same physical line in which the `<carriage-return>` occurred.

The `<carriage-return>` shall be the character designated by ‘\r’ in the C language binding. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the movement to the beginning of the line.

2.2.2.24 character: A sequence of one or more bytes representing a single graphic symbol.

NOTE: This term corresponds in the C Standard {7} to the term *multibyte character*, noting that a single-byte character is a special case of multibyte character. Unlike the usage in the C Standard {7}, *character* here has no necessary relationship with storage space, and *byte* is used when storage space is discussed.

[POSIX.1 {8}]

(See 2.4 for a further explanation of the graphical representations of characters, or “glyphs,” versus character encodings.)

2.2.2.25 character class: A named set of characters sharing an attribute associated with the name of the class.

The classes and the characters that they contain are dependent on the value of the LC_CTYPE category in the current locale; see 2.5.

2.2.2.26 character special file: A file that refers to a device.

One specific type of character special file is a terminal device file, whose access is defined in POSIX.1 {8} section 7.1. Other character special files have no structure defined by this standard, and their use is unspecified by this standard. [POSIX.1 {8}]

2.2.2.27 circumflex: The character “^”.

2.2.2.28 collating element: The smallest entity used to determine the logical ordering of strings.

See *collation sequence* (2.2.2.30). A collating element shall consist of either a single character, or two or more characters collating as a single entity. The value of the LC_COLLATE category in the current locale determines the current set of collating elements.

2.2.2.29 collation: The logical ordering of strings according to defined precedence rules.

These rules identify a collation sequence between the collating elements, and such additional rules that can be used to order strings consisting of multiple collating elements.

2.2.2.30 collation sequence: The relative order of collating elements as determined by the setting of the LC_COLLATE category in the current locale.

The character order, as defined for the LC_COLLATE category in the current locale (see 2.5.2.2), defines the relative order of all collating elements, such that each element occupies a unique position in the order. In addition, one or more collation weights can be assigned for each collating element; these weights are used to determine the relative order of strings in, e.g., the `sort` utility.

Multilevel sorting is accomplished by assigning elements one or more collation weights, up to the limit {COLL_WEIGHTS_MAX}. On each level, elements may be given the same weight (at the primary level, called an *equivalence class*; see 2.2.2.47) or be omitted from the sequence. Strings that collate equal using the first assigned weight (primary ordering), are then compared using the next assigned weight (secondary ordering), and so on.

2.2.2.31 column position: A unit of horizontal measure related to characters in a line. 2

It is assumed that each character in a character set has an intrinsic column width independent of any output device. Each printable character in the portable character set has a column width of one. The standard utilities, when used as described in this standard, assume that all characters have integral column widths. The column width of a character is not necessarily related to the internal representation of the character (numbers of bits or octets). 2

The column position of a character in a line is defined as one plus the sum of the column widths of the preceding characters in the line. Column positions are numbered starting from 1. 2

2.2.2.32 command: A directive to the shell to perform a particular task; see 3.9.

2.2.2.33 current working directory: See *working directory* in 2.2.2.159.

2.2.2.34 command language interpreter: See 2.2.2.133.

2.2.2.35 directory: A file that contains directory entries.

No two directory entries in the same directory shall have the same name. [POSIX.1 {8}]

2.2.2.36 directory entry [link]: An object that associates a filename with a file.

Several directory entries can associate names with the same file. [POSIX.1 {8}]

2.2.2.37 dollar-sign: The character “\$”.

This standard permits the substitution of the “currency symbol” graphic defined in ISO/IEC 646 {1} for this symbol when the character set being used has substituted that graphic for the graphic \$. The graphic symbol \$ is always used in this standard, but not in any monetary sense.

2.2.2.38 dot: The filename consisting of a single dot character (.).

See *pathname resolution* in 2.2.2.104. [POSIX.1 {8}]

In the context of shell special built-in utilities, see 3.14.4.

2.2.2.39 dot-dot: The filename consisting solely of two dot characters (.).

See *pathname resolution* in 2.2.2.104. [POSIX.1 {8}]

2.2.2.40 double-quote: The character “””, also known as *quotation-mark*.

2.2.2.41 effective group ID: An attribute of a process that is used in determining various permissions, including file access permissions, described in 2.2.2.55.

See *group ID*. This value is subject to change during the process lifetime, as described in POSIX.1 {8} 3.1.2 (*exec*) and 4.2.2 [*setgid()*]. [POSIX.1 {8}]

2.2.2.42 effective user ID: An attribute of a process that is used in determining various permissions, including file access permissions.

See *user ID*. This value is subject to change during the process lifetime, as described in POSIX.1 {8} 3.1.2 (*exec*) and 4.2.2 [*setuid()*]. [POSIX.1 {8}]

2.2.2.43 empty directory: A directory that contains, at most, directory entries for dot and dot-dot. [POSIX.1 {8}]

2.2.2.44 empty line: A line consisting of only a <newline> character.

See also *blank line* (2.2.2.17).

2.2.2.45 empty string [null string]: A character array whose first element is a null character. [POSIX.1 {8}]

2.2.2.46 Epoch: The time 0 hours, 0 minutes, 0 seconds, January 1, 1970, Coordinated Universal Time.

See *seconds since the Epoch*. [POSIX.1 {8}]

2.2.2.47 equivalence class: A set of collating elements with the same primary collation weight. 1 1

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

The collation order of elements within an equivalence class is determined by the weights assigned on any subsequent levels after the primary weight. 1 1

2.2.2.48 executable file: A regular file acceptable as a new process image file by the equivalent of the POSIX.1 {8} *exec* family of functions, and thus usable as one form of a utility.

See *exec* in POSIX.1 {8} 3.1.2. The standard utilities described as compilers can produce executable files, but other unspecified methods of producing executable files may also be provided. The internal format of an executable file is unspecified, but a conforming application shall not assume an executable file is a text file.

2.2.2.49 execute: To perform the actions described in 3.9.1.1.

See also *invoke* (2.2.2.79).

2.2.2.50 extended regular expression: A pattern (sequence of characters or symbols) constructed according to the rules defined in 2.8.4.

2.2.2.51 extended security controls: A concept of the underlying system, as follows. [POSIX.1 {8}]

The access control (see *file access permissions*) and privilege (see *appropriate privileges* in 2.2.2.6) mechanisms have been defined to allow implementation-defined extended security controls. These permit an implementation to provide security mechanisms to implement different security policies than described in POSIX.1 {8}. These mechanisms shall not alter or override the defined semantics of any of the functions in POSIX.1 {8}.

2.2.2.52 feature test macro: A `#defined` symbol used to determine whether a particular set of features will be included from a header.

See POSIX.1 {8} 2.7.1. [POSIX.1 {8}]

2.2.2.53 FIFO special file [FIFO]: A type of file with the property that data written to such a file is read on a first-in-first-out basis.

Other characteristics of *FIFOs* are described in POSIX.1 {8} 5.3.1 [*open()*], 6.4.1 [*read()*], 6.4.2 [*write()*], and 6.5.3 [*lseek()*]. [POSIX.1 {8}]

2.2.2.54 file: An object that can be written to, or read from, or both.

A file has certain attributes, including access permissions and type. File types include regular file, character special file, block special file, FIFO special file, and directory. Other types of files may be defined by the implementation. [POSIX.1 {8}]

2.2.2.55 file access permissions: A concept of the underlying system, as follows. [POSIX.1 {8}]

The standard file access control mechanism uses the file permission bits, as described below. These bits are set at file creation by *open()*, *creat()*, *mkdir()*, and *mkfifo()* and are changed by *chmod()*. These bits are read by *stat()* or *fstat()*.

Implementations may provide *additional* or *alternate* file access control mechanisms, or both. An additional access control mechanism shall only further restrict the access permissions defined by the file permission bits. An alternate access control mechanism shall:

- (1) Specify file permission bits for the file owner class, file group class, and file other class of the file, corresponding to the access permissions, to be returned by *stat()* or *fstat()*.
- (2) Be enabled only by explicit user action, on a per-file basis by the file owner or a user with the appropriate privilege.

- (3) Be disabled for a file after the file permission bits are changed for that file with *chmod()*. The disabling of the alternate mechanism need not disable any additional mechanisms defined by an implementation.

Whenever a process requests file access permission for read, write, or execute/search, if no additional mechanism denies access, access is determined as follows:

- (1) If a process has the appropriate privilege:
 - (a) If read, write, or directory search permission is requested, access is granted.
 - (b) If execute permission is requested, access is granted if execute permission is granted to at least one user by the file permission bits or by an alternate access control mechanism; otherwise, access is denied.
- (2) Otherwise:
 - (a) The file permission bits of a file contain read, write, and execute/search permissions for the file owner class, file group class, and file other class.
 - (b) Access is granted if an alternate access control mechanism is not enabled and the requested access permission bit is set for the class (file owner class, file group class, or file other class) to which the process belongs, or if an alternate access control mechanism is enabled and it allows the requested access; otherwise, access is denied.

2.2.2.56 file descriptor: A per-process unique, nonnegative integer used to identify an open file for the purpose of file access. [POSIX.1 {8}]

2.2.2.57 file group class: The property of a file indicating access permissions for a process related to the process's group identification.

A process is in the file group class of a file if the process is not in the file owner class and if the effective group ID or one of the supplementary group IDs of the process matches the group ID associated with the file. Other members of the class may be implementation defined. [POSIX.1 {8}]

2.2.2.58 file hierarchy: A concept of the underlying system, as follows. [POSIX.1 {8}]

Files in the system are organized in a hierarchical structure in which all of the nonterminal nodes are directories and all of the terminal nodes are any other type of file. Because multiple directory entries may refer to the same file, the hierarchy is properly described as a "directed graph."

2.2.2.59 file mode: An object containing the file permission bits and other characteristics of a file, as described in POSIX.1 {8} 5.6.1. [POSIX.1 {8}]

2.2.2.60 file mode bits: A file's file permission bits, set-user-ID-on-execution bit (S_ISUID), and set-group-ID-on-execution bit (S_ISGID) (see POSIX.1 {8} 5.6.1.2).

2.2.2.61 filename: A name consisting of 1 to {NAME_MAX} bytes used to name a file.

The characters composing the name may be selected from the set of all character values excluding the slash character and the null character. The filenames dot and dot-dot have special meaning; see *pathname resolution* in 2.2.2.104. A filename is sometimes referred to as a pathname component. [POSIX.1 {8}]

2.2.2.62 filename portability: A concept of the underlying system, as follows. [POSIX.1 {8}]

Filenames should be constructed from the portable filename character set because the use of other characters can be confusing or ambiguous in certain contexts.

2.2.2.63 file offset: The byte position in the file where the next I/O operation begins.

Each open file description associated with a regular file, block special file, or directory has a file offset. A character special file that does not refer to a terminal device may have a file offset. There is no file offset specified for a pipe or FIFO. [POSIX.1 {8}]

2.2.2.64 file other class: The property of a file indicating access permissions for a process related to the process's user and group identification.

A process is in the file other class of a file if the process is not in the file owner class or file group class. [POSIX.1 {8}]

2.2.2.65 file owner class: The property of a file indicating access permissions for a process related to the process's user identification.

A process is in the file owner class of a file if the effective user ID of the process matches the user ID of the file. [POSIX.1 {8}]

2.2.2.66 file permission bits: Information about a file that is used, along with other information, to determine if a process has read, write, or execute/search permission to a file.

The bits are divided into three parts: owner, group, and other. Each part is used with the corresponding file class of processes. These bits are contained in the file mode, as described in POSIX.1 {8} 5.6.1. The detailed usage of the file permission bits in access decisions is described in *file access permissions* in 2.2.2.55. [POSIX.1 {8}]

2.2.2.67 file serial number: A per-file-system unique identifier for a file.

File serial numbers are unique throughout a file system. [POSIX.1 {8}]

2.2.2.68 file system: A collection of files and certain of their attributes.

It provides a name space for file serial numbers referring to those files.
[POSIX.1 {8}]

2.2.2.69 file times update: A concept of the underlying system, as follows.
[POSIX.1 {8}]

Each file has three distinct associated time values: *st_atime*, *st_mtime*, and *st_ctime*. The *st_atime* field is associated with the times that the file data is accessed; *st_mtime* is associated with the times that the file data is modified; and *st_ctime* is associated with the times that file status is changed. These values are returned in the file characteristics structure, as described in POSIX.1 {8} 5.6.1.

Any function in this standard that is required to read or write file data or change the file status indicates which of the appropriate time-related fields are to be “marked for update.” If an implementation of such a function marks for update a time-related field not specified by this standard, this shall be documented, except that any changes caused by pathname resolution need not be documented. For the other functions in this standard (those that are not explicitly required to read or write file data or change file status, but that in some implementations happen to do so), the effect is unspecified.

An implementation may update fields that are marked for update immediately, or it may update such fields periodically. When the fields are updated, they are set to the current time and the update marks are cleared. All fields that are marked for update shall be updated when the file is no longer open by any process, or when a *stat()* or *fstat()* is performed on the file. Other times at which updates are done are unspecified. Updates are not done for files on read-only file systems.

2.2.2.70 file type: See *file* in 2.2.2.54.

2.2.2.71 filter: A command whose operation consists of reading data from standard input or a list of input files and writing data to standard output.

Typically, its function is to perform some transformation on the data stream.

2.2.2.72 foreground process: A process that is a member of a foreground process group. [POSIX.1 {8}]

2.2.2.73 foreground process group: A process group whose member processes have certain privileges, denied to processes in background process groups, when accessing their controlling terminal.

Each session that has established a connection with a controlling terminal has exactly one process group of the session as the foreground process group of that

controlling terminal. See POSIX.1 {8} 7.1.1.4. [POSIX.1 {8}]

2.2.2.74 <form-feed>: A character that in the output stream shall indicate that printing should start on the next page of an output device. 1

The <form-feed> shall be the character designated by ‘\f’ in the C language binding. If <form-feed> is not the first character of an output line, the result is unspecified. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the movement to the next page.

2.2.2.75 group ID: A nonnegative integer, which can be contained in an object of type *gid_t*, that is used to identify a group of system users.

Each system user is a member of at least one group. When the identity of a group is associated with a process, a group ID value is referred to as a real group ID, an effective group ID, one of the (optional) supplementary group IDs, or an (optional) saved set-group-ID. [POSIX.1 {8}]

2.2.2.76 hard link: The relationship between two directory entries that represent the same file; the result of an execution of the *ln* utility or the POSIX.1 {8} *link()* function.

2.2.2.77 home directory: The current directory associated with a user at the time of login.

2.2.2.78 incomplete line: A sequence of text consisting of one or more non-<newline> characters at the end of the file.

2.2.2.79 invoke: To perform the actions described in 3.9.1.1, except that searching for shell functions and special built-ins is suppressed.

See also *execute* (2.2.2.49).

2.2.2.80 job control: A facility that allows users to selectively stop (suspend) the execution of processes and continue (resume) their execution at a later point.

The user typically employs this facility via the interactive interface jointly supplied by the terminal I/O driver and a command interpreter. POSIX.1 {8} conforming implementations may optionally support job control facilities; the presence of this option is indicated to the application at compile time or run time by the definition of the `{_POSIX_JOB_CONTROL}` symbol; see POSIX.1 {8} 2.9. [POSIX.1 {8}]

2.2.2.81 line: A sequence of text consisting of zero or more non-<newline> characters plus a terminating <newline> character.

- 620 **2.2.2.82 link:** See *directory entry* in 2.2.2.36.
- 621 **2.2.2.83 link count:** The number of directory entries that refer to a particular
622 file. [POSIX.1 {8}]
- 623 **2.2.2.84 locale:** The definition of the subset of a user's environment that
624 depends on language and cultural conventions; see 2.5.
- 625 **2.2.2.85 login:** The unspecified activity by which a user gains access to the
626 system.
627 Each login shall be associated with exactly one login name. [POSIX.1 {8}]
- 628 **2.2.2.86 login name:** A user name that is associated with a login. [POSIX.1 {8}]
- 629 **2.2.2.87 mode:** A collection of attributes that specifies a file's type and its access
630 permissions.
631 See *file access permissions* in 2.2.2.55. [POSIX.1 {8}]
- 632 **2.2.2.88 multicharacter collating element:** A sequence of two or more charac-
633 ters that collate as an entity.
634 For example, in some coded character sets, an accented character is represented
635 by a (nonspacing) accent, followed by the letter. Another example is the Spanish
636 elements "ch" and "ll."
- 637 **2.2.2.89 negative response:** An input string that matches one of the responses
638 acceptable to the LC_MESSAGES category keyword `noexpr`, matching an
639 extended regular expression in the current locale.
640 See 2.5.
- 641 **2.2.2.90 <newline>:** A character that in the output stream shall indicate that 1
642 printing should start at the beginning of the next line.
643 The <newline> shall be the character designated by '`\n`' in the C language
644 binding. It is unspecified whether this character is the exact sequence transmit-
645 ted to an output device by the system to accomplish the movement to the next
646 line.
- 647 **2.2.2.91 NUL:** A character with all bits set to zero.
- 648 **2.2.2.92 null string:** See *empty string* in 2.2.2.45.

2.2.2.93 number-sign: The character “#”.

This standard permits the substitution of the “pound sign” graphic defined in ISO/IEC 646 {1} for this symbol when the character set being used has substituted that graphic for the graphic #. The graphic symbol # is always used in this standard.

2.2.2.94 object file: A regular file containing the output of a compiler, formatted as input to a linkage editor for linking with other object files into an executable form.

The methods of linking are unspecified and may involve the dynamic linking of objects at run-time. The internal format of an object file is unspecified, but a conforming application shall not assume an object file is a text file.

2.2.2.95 open file: A file that is currently associated with a file descriptor. [POSIX.1 {8}]

2.2.2.96 operand: An argument to a command that is generally used as an object supplying information to a utility necessary to complete its processing.

Operands generally follow the options in a command line. See 2.10.1.

2.2.2.97 option: An argument to a command that is generally used to specify changes in the *utility*’s default behavior; see 2.10.1.

2.2.2.98 option-argument: A parameter that follows certain options.

In some cases an option-argument is included within the same argument string as the option; in most cases it is the next argument. See 2.10.1.

2.2.2.99 parent directory:

- (1) When discussing a given directory, the directory that both contains a directory entry for the given directory and is represented by the path-name dot-dot in the given directory.
- (2) When discussing other types of files, a directory containing a directory entry for the file under discussion.

This concept does not apply to dot and dot-dot. [POSIX.1 {8}]

2.2.2.100 parent process: See *process* in 2.2.2.114. [POSIX.1 {8}]

2.2.2.101 parent process ID: An attribute of a new process after it is created by a currently active process.

The parent process ID of a process is the process ID of its creator, for the lifetime of the creator. After the creator’s lifetime has ended, the parent process ID is the process ID of an implementation-defined system process. [POSIX.1 {8}]

2.2.2.102 pathname: A string that is used to identify a file.

A pathname consists of, at most, {PATH_MAX} bytes, including the terminating null character. It has an optional beginning slash, followed by zero or more filenames separated by slashes. If the pathname refers to a directory, it may also have one or more trailing slashes. Multiple successive slashes are considered to be the same as one slash. A pathname that begins with two successive slashes may be interpreted in an implementation-defined manner, although more than two leading slashes shall be treated as a single slash. The interpretation of the pathname is described in *pathname resolution* in 2.2.2.104. [POSIX.1 {8}]

2.2.2.103 pathname component: See *filename* in 2.2.2.61. [POSIX.1 {8}]

2.2.2.104 pathname resolution: A concept of the underlying system, as follows. [POSIX.1 {8}]

Pathname resolution is performed for a process to resolve a pathname to a particular file in a file hierarchy. There may be multiple pathnames that resolve to the same file.

Each filename in the pathname is located in the directory specified by its predecessor (for example, in the pathname fragment “a/b”, file “b” is located in directory “a”). Pathname resolution fails if this cannot be accomplished. If the pathname begins with a slash, the predecessor of the first filename in the pathname is taken to be the root directory of the process (such pathnames are referred to as absolute pathnames). If the pathname does not begin with a slash, the predecessor of the first filename of the pathname is taken to be the current working directory of the process (such pathnames are referred to as “relative pathnames”).

The interpretation of a pathname component is dependent on the values of {NAME_MAX} and {_POSIX_NO_TRUNC} associated with the path prefix of that component. If any pathname component is longer than {NAME_MAX}, and {_POSIX_NO_TRUNC} is in effect for the path prefix of that component [see *pathconf()* in POSIX.1 {8} 5.7.1], the implementation shall consider this an error condition. Otherwise, the implementation shall use the first {NAME_MAX} bytes of the pathname component.

The special filename dot refers to the directory specified by its predecessor. The special filename dot-dot refers to the parent directory of its predecessor directory. As a special case, in the root directory, dot-dot may refer to the root directory itself.

A pathname consisting of a single slash resolves to the root directory of the process. A null pathname is invalid.

2.2.2.105 path prefix: A pathname, with an optional ending slash, that refers to a directory. [POSIX.1 {8}]

2.2.2.106 pattern: A sequence of characters used either with regular expression notation (see 2.8) or for pathname expansion (see 3.6.6), as a means of selecting various character strings or pathnames, respectively.

The syntaxes of the two patterns are similar, but not identical; this standard always indicates the type of pattern being referred to in the immediate context of the use of the term.

2.2.2.107 period: The character “.”.

The term *period* is contrasted against *dot* (2.2.2.38), which is used to describe a specific directory entry.

2.2.2.108 permissions: See *file access permissions* in 2.2.2.55.

2.2.2.109 pipe: An object accessed by one of the pair of file descriptors created by the POSIX.1 {8} *pipe()* function.

Once created, the file descriptors can be used to manipulate it, and it behaves identically to a FIFO special file when accessed in this way. It has no name in the file hierarchy. [POSIX.1 {8}]

2.2.2.110 portable character set: The set of characters described in 2.4 that is supported on all conforming systems.

This term is contrasted against the smaller *portable filename character set*; see 2.2.2.111.

2.2.2.111 portable filename character set: The set of characters from which portable filenames are constructed.

For a filename to be portable across conforming implementations of this standard, it shall consist only of the following 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
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -
```

The last three characters are the period, underscore, and hyphen characters, respectively. The hyphen shall not be used as the first character of a portable filename. Upper- and lowercase letters shall retain their unique identities between conforming implementations. In the case of a portable pathname, the slash character may also be used. [POSIX.1 {8}]

2.2.2.112 printable character: One of the characters included in the *print* character classification of the *LC_CTYPE* category in the current locale; see 2.5.2.1.

2.2.2.113 privilege: See *appropriate privileges* in 2.2.2.6. [POSIX.1 {8}]

2.2.2.114 process: An address space and single thread of control that executes within that address space, and its required system resources.

A process is created by another process issuing the POSIX.1 {8} *fork()* function. The process that issues *fork()* is known as the parent process, and the new process created by the *fork()* is known as the child process. [POSIX.1 {8}]

The attributes of processes required by POSIX.2 form a subset of those in POSIX.1 {8}; see 2.9.1.

2.2.2.115 process group: A collection of processes that permits the signaling of related processes.

Each process in the system is a member of a process group that is identified by a process group ID. A newly created process joins the process group of its creator. [POSIX.1 {8}]

2.2.2.116 process group ID: The unique identifier representing a process group during its lifetime.

A process group ID is a positive integer that can be contained in a *pid_t*. It shall not be reused by the system until the process group lifetime ends. [POSIX.1 {8}]

2.2.2.117 process group leader: A process whose process ID is the same as its process group ID. [POSIX.1 {8}]

2.2.2.118 process ID: The unique identifier representing a process.

A process ID is a positive integer that can be contained in a *pid_t*. A process ID shall not be reused by the system until the process lifetime ends. In addition, if there exists a process group whose process group ID is equal to that process ID, the process ID shall not be reused by the system until the process group lifetime ends. A process that is not a system process shall not have a process ID of 1. [POSIX.1 {8}]

2.2.2.119 program: A prepared sequence of instructions to the system to accomplish a defined task.

The term *program* in POSIX.2 encompasses applications written in the Shell Command Language, complex utility input languages (for example, *awk*, *lex*, *sed*, etc.), and high-level languages.

2.2.2.120 read-only file system: A file system that has implementation-defined characteristics restricting modifications. [POSIX.1 {8}]

2.2.2.121 real group ID: The attribute of a process that, at the time of process creation, identifies the group of the user who created the process.

See *group ID* in 2.2.2.75. This value is subject to change during the process lifetime, as described in POSIX.1 {8} 4.2.2 [*setgid()*]. [POSIX.1 {8}]

2.2.2.122 real user ID: The attribute of a process that, at the time of process creation, identifies the user who created the process.

See *user ID* in 2.2.2.154. This value is subject to change during the process lifetime, as described in POSIX.1 {8} 4.2.2 [*setuid()*]. [POSIX.1 {8}]

2.2.2.123 regular expression: A pattern (sequence of characters or symbols) constructed according to the rules defined in 2.8. 1

2.2.2.124 regular file: A file that is a randomly accessible sequence of bytes, with no further structure imposed by the system. [POSIX.1 {8}]

2.2.2.125 relative pathname: See *pathname resolution* in 2.2.2.104. [POSIX.1 {8}]

2.2.2.126 root directory: A directory, associated with a process, that is used in pathname resolution for pathnames that begin with a slash. [POSIX.1 {8}]

2.2.2.127 saved set-group-ID: An attribute of a process that allows some flexibility in the assignment of the effective group ID attribute, when the saved set-user-ID option is implemented, as described in POSIX.1 {8} 3.1.2 (*exec*) and 4.2.2 [*setgid()*]. [POSIX.1 {8}]

2.2.2.128 saved set-user-ID: An attribute of a process that allows some flexibility in the assignment of the effective user ID attribute, when the saved set-user-ID option is implemented, as described in POSIX.1 {8} 3.1.2 and 4.2.2 [*setuid()*]. [POSIX.1 {8}]

2.2.2.129 seconds since the Epoch: A value to be interpreted as the number of seconds between a specified time and the Epoch.

A Coordinated Universal Time name (specified in terms of seconds (*tm_sec*), minutes (*tm_min*), hours (*tm_hour*), days since January 1 of the year (*tm_yday*), and calendar year minus 1900 (*tm_year*) is related to a time represented as seconds since the Epoch, according to the expression below.

If the year < 1970 or the value is negative, the relationship is undefined. If the year ≥ 1970 and the value is nonnegative, the value is related to a Coordinated Universal Time name according to the expression:

821 $tm_sec + tm_min*60 + tm_hour*3\,600 + tm_yday*86\,400 +$
 822 $(tm_year-70)*31\,536\,000 + ((tm_year-69)/4)*86\,400$

823 [POSIX.1 {8}]

824 **2.2.2.130 session:** A collection of process groups established for job control
 825 purposes.

826 Each process group is a member of a session. A process is considered to be a
 827 member of the session of which its process group is a member. A newly created
 828 process joins the session of its creator. A process can alter its session membership
 829 (see POSIX.1 {8} 4.3.2 [setsid()]). Implementations that support the POSIX.1 {8}
 830 setpgid() function (see POSIX.1 {8} 4.3.3) can have multiple process groups in the
 831 same session. [POSIX.1 {8}]

832 **2.2.2.131 session leader:** A process that has created a session; see POSIX.1 {8}
 833 4.3.2 [setsid()]. [POSIX.1 {8}]

834 **2.2.2.132 session lifetime:** The period between when a session is created and
 835 the end of the lifetime of all the process groups that remain as members of the
 836 session. [POSIX.1 {8}]

837 **2.2.2.133 shell:** A program that interprets sequences of text input as commands.

838 It may operate on an input stream or it may interactively prompt and read com-
 839 mands from a terminal.

840 **2.2.2.134 Shell, The:** The Shell Command Language Interpreter (see 4.56), a
 841 specific instance of a shell.

842 **2.2.2.135 shell script:** A file containing shell commands.

843 If the file is made executable, it can be executed by specifying its name as a sim-
 844 ple command (see the description of *simple command* in 3.9.1). Execution of a
 845 shell script causes a shell to execute the commands within the script. Alternately,
 846 a shell can be requested to execute the commands in a shell script by specifying
 847 the name of the shell script as the operand to the sh utility.

848 **2.2.2.136 signal:** A mechanism by which a process may be notified of, or affected
 849 by, an event occurring in the system.

850 Examples of such events include hardware exceptions and specific actions by
 851 processes. The term *signal* is also used to refer to the event itself. [POSIX.1 {8}]

852 **2.2.2.137 single-quote:** The character “'”, also known as *apostrophe*.

2.2.2.138 slash: The character “/”, also known as *solidus*.

2.2.2.139 source code: When dealing with the Shell Command Language, source code is input to the command language interpreter.

The term *shell script* is synonymous with this meaning.

When dealing with the C Language Bindings Option, source code is input to a compiler conforming to the C Standard {7}.

When dealing with another ISO/IEC conforming language, source code is input to a compiler conforming to that ISO/IEC standard.

Source code also refers to the input statements prepared for the following standard utilities: *awk*, *bc*, *ed*, *lex*, *localedef*, *make*, *sed*, and *yacc*.

Source code can also refer to a collection of sources meeting any or all of these meanings.

2.2.2.140 <space>: The character defined in 2.4 as <space>.

The <space> character is a member of the *space* character class of the current locale, but represents the single character, and not all of the possible members of the class. (See 2.2.2.158.)

2.2.2.141 standard error: An output stream usually intended to be used for diagnostic messages.

2.2.2.142 standard input: An input stream usually intended to be used for primary data input.

2.2.2.143 standard output: An output stream usually intended to be used for primary data output.

2.2.2.144 standard utilities: The utilities defined by this standard, in the Sections 4, 5, and 6, and Annex A, and Annex C, and in similar sections of utility definitions introduced in future revisions of, and supplements to, this standard.

2.2.2.145 stream: An ordered sequence of characters, as described by the C Standard {7}.

2.2.2.146 supplementary group ID: An attribute of a process used in determining file access permissions.

A process has up to {NGROUPS_MAX} supplementary group IDs in addition to the effective group ID. The supplementary group IDs of a process are set to the supplementary group IDs of the parent process when the process is created. Whether a process’s effective group ID is included in or omitted from its list of supplementary group IDs is unspecified. [POSIX.1 {8}]

- 887 **2.2.2.147 system:** An implementation of this standard.
- 888 **2.2.2.148 <tab>:** The horizontal tab character.
- 889 **2.2.2.149 terminal [terminal device]:** A character special file that obeys the
890 specifications of the POSIX.1 {8} General Terminal Interface. [POSIX.1 {8}]
- 891 **2.2.2.150 text column:** A roughly rectangular block of characters capable of
892 being laid out side-by-side next to other text columns on an output page or termi-
893 nal screen.
- 894 The widths of text columns are measured in column positions.
- 895 **2.2.2.151 text file:** A file that contains characters organized into one or more
896 lines.
- 897 The lines shall not contain NUL characters and none shall exceed {LINE_MAX}
898 bytes in length, including the <newline>. Although POSIX.1 {8} does not distin-
899 guish between text files and binary files (see the C Standard {7}), many utilities
900 only produce predictable or meaningful output when operating on text files. The
901 standard utilities that have such restrictions always specify *text files* in their
902 Standard Input or Input Files subclauses.
- 903 **2.2.2.152 tilde:** The character “~”.
- 904 **2.2.2.153 user database:** See Section 9 in POSIX.1 {8}.
- 905 **2.2.2.154 user ID:** A nonnegative integer, which can be contained in an object of
906 type *uid_t*, that is used to identify a system user.
- 907 When the identity of a user is associated with a process, a user ID value is
908 referred to as a real user ID, an effective user ID, or an (optional) saved
909 set-user-ID. [POSIX.1 {8}]
- 910 **2.2.2.155 user name:** A string that is used to identify a user, as described in
911 POSIX.1 {8} 9.1. [POSIX.1 {8}]
- 912 **2.2.2.156 utility:** A program that can be called by name from a shell to perform
913 a specific task, or related set of tasks.
- 914 This program shall either be an executable file, such as might be produced by a
915 compiler/linker system from computer source code, or a file of shell source code,
916 directly interpreted by the shell. The program may have been produced by the
917 user, provided by the implementor of this standard, or acquired from an indepen-
918 dent distributor. The term *utility* does not apply to the special built-in utilities
919 provided as part of the shell command language; see 3.14. The system may imple-
920 ment certain utilities as shell functions (see 3.9.5) or built-ins (see 2.3), but only

an application that is aware of the command search order described in 3.9.1.1 or of performance characteristics can discern differences between the behavior of such a function or built-in and that of a true executable file.

2.2.2.157 <vertical-tab>: The vertical tab character.

2.2.2.158 white space: A sequence of one or more characters that belong to the space character class as defined via the LC_CTYPE category in the current locale.

In the POSIX Locale, white space consists of one or more <blank>s (<space>s and <tab>s), <newline>s, <carriage-return>s, <form-feed>s, and <vertical-tab>s.

2.2.2.159 working directory [current working directory]: A directory, associated with a process, that is used in pathname resolution for pathnames that do not begin with a slash.

2.2.2.160 write: To output characters to a file, such as standard output or standard error.

Unless otherwise stated, standard output is the default output destination for all uses of the term *write*.

2.2.2.161 General Terms Rationale. *(This subclause is not a part of P1003.2)*

Many of the terms originated in POSIX.1 {8} and are duplicated in this standard to meet editorial requirements. In some cases, there is supplementary text that presents additional information concerning POSIX.2 aspects of the concept.

This standard uses the term *character* to mean a sequence of one or more bytes representing a single graphic symbol, as defined in POSIX.1 {8}. The deviation in the exact text of the C Standard {7} definition for *byte* meets the intent of the C Standard {7} Rationale and the developers of POSIX.1 {8}, but clears up the ambiguity raised by the term *basic execution character set*, which is not defined in POSIX.1 {8}. It is expected that a future version of POSIX.1 {8} will align with the text used here. The octet-minimum requirement is merely a reflection of the {CHAR_BIT} value in POSIX.1 {8} and the C Standard {7}.

The POSIX.1 {8} term *file mode* is a superset of the POSIX.2 *file mode bits*. POSIX.1 {8} defines the file mode as the entire *mode_t* object (which includes the file type in historically the upper four bits, the sticky bit on most implementations, and potentially other nonstandardized attributes), while POSIX.2 file mode bits include only the eleven defined bits.

The terms *command* and *utility* are related but have distinct meanings. *Command* is defined as “a directive to a shell to perform a specific task.” The directive can be in the form of a single utility name (for example, *ls*), or the directive can take the form of a compound command (for example, *ls | grep name | pr*).

A utility is a program that is callable by name from a shell. Issuing only the utility's name to a shell is the equivalent of a one-word command. A utility may be invoked as a separate program that executes in a different process than the command language interpreter, or may be implemented as a part of the command language interpreter. For example, the `echo` command (the directive to perform a specific task) may be implemented such that the `echo` utility (the logic that performs the task of echoing) is in a separate program; and therefore, is executed in a process that is different than the command language interpreter. Conversely, the logic that performs the `echo` utility could be built into the command language interpreter; and therefore, execute in the same process as the command language interpreter.

The terms *tool* and *application* can be thought of as being synonymous with *utility* from the perspective of the operating system kernel. Tools, applications, and utilities have historically run, typically, in processes above the kernel level. Tools and utilities have been historically a part of the operating system nonkernel code, and performed system related functions such as listing directory contents, checking file systems, repairing file systems, or extracting system status information. Applications have not generally been a part of the operating system, and perform nonsystem related functions such as word processing, architectural design, mechanical design, workstation publishing, or financial analysis. Utilities have most frequently been provided by the operating system vendor, applications by third party software vendors or by the users themselves. Nevertheless, the standard does not differentiate between tools, utilities, and applications when it comes to receiving services from the system, a shell, or the standard utilities. (For example, the `xargs` utility invokes another utility; it would be of fairly limited usefulness if the users couldn't run their own applications in place of the standard utilities.) Utilities are not applications in the sense that they are not themselves subjects to the restrictions of this standard or any other standard—there is no requirement for `grep`, `stty`, or any of the utilities defined here to be any of the classes of Conforming POSIX.2 Applications.

The term *text file* does not prevent the inclusion of control or other nonprintable characters (other than NUL). Therefore, standard utilities that list text files as inputs or outputs are either able to process the special characters gracefully or they explicitly describe their limitations within their individual subclauses. The definition of *text file* has caused a good deal of controversy. The only difference between text and binary here is that text files have lines of (less than {`LINE_MAX`}) bytes, with no NUL characters, each terminated by a `<newline>` character. The definition allows a file with a single `<newline>`, but not a totally empty file, to be called a text file. If a file ends with an incomplete line it is not strictly a text file by this definition. A related point is that the `<newline>` character referred to in this standard is not some generic line separator, but a single character; files created on systems where they use multiple characters for ends of lines are not portable to all POSIX systems without some translation process unspecified by this standard.

The term *hard link* is historically-derived. In systems without extensions to `ln`, it is a synonym for *link*. The concept of a *symbolic link* originated with BSD systems and the term *hard* is used to differentiate between the two types of links.

There are some terms used that are undefined in POSIX.2, POSIX.1 {8}, or the C Standard {7}. The working group believes that these terms have a “common usage,” and that a definition in POSIX.2 would not be appropriate. Terms in this category include, but are not limited to, the following: *application*, *character set*, *login session*, *user*. Good sources for general terms of this type are the *ISO/AFNOR Dictionary of Computer Science* {B12} and *IEEE Dictionary* {B18}.

The term *file name* was defined in previous drafts to be a synonym for *pathname*. It was removed in the face of objections that it was too close to *filename*, which means something different (a pathname component). The general solution to this has been to use the term *file* in parameter names, rather than *file_name*, and to make more liberal use of the correct term, *pathname*; an alternate solution has been to replace *file name* with *the name of the file*.

Many character names are included in this subclause. Because of historical usage, some of these names are a bit different than the ones used in international standards for character sets, such as ISO/IEC 646 {1}. It was felt that many more UNIX system people than character set lawyers would be reading and reviewing the standard, so the former group was the one accommodated. On the other hand, the precise definitions of <space>, <blank>, and *white space* have replaced common usage (where they have been used virtually interchangeably), as the standard attempts to balance readability against precision.

In earlier drafts, the names for the character pairs (), [], and { } were referred to as “opening” and “closing” parentheses, brackets, and braces. These were changed to the current “left” and right.” When the characters are used to express natural language, the terms “open” and “close” imply text direction more strongly than “left” and “right.” By POSIX.2 definition, the character <open-parenthesis> will always be mapped to the glyph ‘(’ regardless of the locale. But when reading right-to-left, the opening punctuation of a parenthesized text segment would be ‘)’. The <left-parenthesis> and <right-parenthesis> forms are the correct ones because the punctuation appears on the left and right, respectively, of the parenthesized text regardless of the direction one might be reading the text.

The <backspace> character and the ERASE special character defined in POSIX.1 {8} should not be confused. The use of the <backspace> character and the ERASE special character defined in the POSIX.1 {8} *termios* clause on special characters (7.1.1.9) are distinct even though the ERASE special character may be set to <backspace>.

In most one-byte character sets, such as ASCII, the concepts of column positions is identical to character positions and to bytes. Therefore, it has been historically acceptable for some implementations to describe line folding or tab stops or table column alignment in terms of bytes or character positions. Other character sets pose complications, as they can have internal representations longer than one octet and they can have displayable characters that have different widths on the terminal screen or printer.

In this standard the term *column positions* has been defined to mean character—not byte—positions in input files (such as “column position 7 of the FORTRAN

input”). Output files describe the column position in terms of the display width of the narrowest printable character in the character set, adjusted to fit the characteristics of the output device. It is very possible that n column positions will not be able to hold n characters in some character sets, unless all of those characters are of the narrowest width. It is assumed that the implementation is aware of the width of the various characters, deriving this information from the value of **LC_CTYPE**, and thus can determine how many column positions to allot for each character in those utilities where it is important. This information is not available to the portable application writer because POSIX.2 provides no interface specification to retrieve such information.

The term *column position* was used instead of the more natural *column* as the latter is frequently used in the standard in the different contexts of columns of figures, columns of table values, etc. Wherever confusion might result, these latter types of columns are referred to as *text columns*.

The definition of *binary file* was removed, as the term is not used in the standard.

The ISO/IEC 646 {1} character set standard permits substitution of national currency symbols for the character \$ in the “reference character set” (which is the same as ASCII). This standard permits the substitution only of the actual characters shown in ISO/IEC 646 {1}: currency sign for the dollar sign and pound sign for the number sign. This document uses the latter names and their symbols, but it is valid for an implementation to accept, for instance, the pound sign (£) as a comment character in the shell, if that is what the locale’s character set uses instead of the number sign (#). Other variation of national currency symbols are not allowed, per the request of the WG15 POSIX working group.

The term *stream* is not related to System V’s STREAMS communications facility; it is derived from historical UNIX system usage and has been made official by the C Standard {7}. The POSIX.2 standard makes no differentiation between C’s *text stream* and *binary stream*.

The formula used in the POSIX.1 {8} definition of *seconds since the Epoch* is not perfect in all cases. See the related rationale in POSIX.1 {8}.

1
1

2.2.3 Abbreviations

For the purposes of this standard, the following abbreviations apply:

2.2.3.1 C Standard: ISO/IEC 9899: ..., *Information processing systems—Programming languages—C* {7}.

2.2.3.2 ERE: An Extended Regular Expression, as defined in 2.8.4.

2.2.3.3 LC_*: An abbreviation used to represent all of the environment variables named in 2.6 whose names begin with the characters “LC_”.

2.2.3.4 POSIX.1: ISO/IEC 9945-1: 1990: *Information technology—Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C Language]* {8}.

2.2.3.5 POSIX.2: This standard.

2.2.3.6 RE [BRE]: A Basic Regular Expression, as defined in 2.8.3.

2.3 Built-in Utilities

Any of the standard utilities may be implemented as *regular built-in* utilities within the command language interpreter. This is usually done to increase the performance of frequently-used utilities or to achieve functionality that would be more difficult in a separate environment. The utilities named in Table 2-2 are frequently provided in built-in form. All of the utilities named in the table have special properties in terms of command search order within the shell, as described in 3.9.1.1.

Table 2-2 – Regular Built-in Utilities

cd	false	kill	true	wait
command	getopts	read	umask	

However, all of the standard utilities, including the regular built-ins in the table, but not the special built-ins described in 3.14, shall be implemented in a manner so that they can be accessed via the POSIX.1 {8} *exec* family of functions (if the underlying operating system provides the services of such a family to application programs) and can be invoked directly by those standard utilities that require it (*env*, *find*, *nohup*, *xargs*).

Since versions shall be provided for all utilities except for those listed previously, an application running on a system that conforms to both POSIX.1 {8} and Section 7 of this standard can use the *exec* family of functions, in addition to the shell command interface in 7.1 [such as the *system()* and *popen()* functions in the C binding] defined by this standard, to execute any of these utilities.

2.3.1 Built-in Utilities Rationale. *(This subclause is not a part of P1003.2)*

In earlier drafts, the table of built-ins implied two things to a conforming application: these may be built-ins and these need not be executable. The second implication has now been removed and all utilities can be *exec*-ed. There is no requirement that these be actually built into the shell itself, but many shells will want to do so because 3.9.1.1 requires that they be found prior to the **PATH** search. The shell could satisfy its requirements by keeping a list of the names and directly accessing the file-system versions regardless of **PATH**. Providing all of the required functionality for those such as *cd* or *read* would be more difficult.

There were originally three justifications for allowing the omission of *exec*-able versions:

- (1) This would require wasting space in the file system, at the expense of very small systems. However, it has been pointed out that all nine in the table can be provided with nine links to a single-line shell script:

1130 \$0 "\$@"

1131 (2) There is no sense in requiring invocation of utilities like `cd` because they
1132 have no value outside the shell environment or cannot be useful in a
1133 child process. However, counter-examples always seemed to be available
1134 for even the strangest cases:

1135 `find . -type d -exec cd {} ; -exec foo {} ;`
1136 (which invokes `foo` on accessible directories)

1137 `ps ... | sed ... | xargs kill`

1138 `find . -exec true ; -a ...`
1139 (where `true` is used for temporary debugging)

1140 (3) It is confusing to have something such as `kill` that can easily be in the
1141 file system in the base standard, but requires built-in status for the UPE
1142 (for the `%` job control job ID notation). It was decided that it was more
1143 appropriate to describe the required functionality (rather than the imple-
1144 mentation) to the system implementors and let them decide how to
1145 satisfy it.

1146 On the other hand, there were objections raised during balloting that any distinc-
1147 tion like this between utilities was not useful to applications and that the cost to
1148 correct it was small. These arguments were ultimately the most effective.

1149 There were varying reasons for including utilities in the table of built-ins:

1150 `cd`, `getopts`, `read`, `umask`, `wait`

1151 The functionality of these utilities is performed more simply
1152 within the context of the current process. An example can be
1153 taken from the usage of the `cd` utility. The purpose of the utility
1154 is to change the working directory for subsequent operations. The
1155 actions of `cd` affect the process in which `cd` is executed and all
1156 subsequent child processes of that process. Based on the
1157 POSIX.1 {8} process model, changes in the process environment of
1158 a child process have no effect on the parent process. If the `cd`
1159 utility were executed from a child process, the working directory
1160 change would be effective only in the child process. Child
1161 processes initiated subsequent to the child process that executed
1162 the `cd` utility would not have a changed working directory rela-
1163 tive to the parent process.

1164 `command` This utility was placed in the table primarily to protect scripts
1165 that are concerned about their **PATH** being manipulated. The
1166 “secure” shell script example in 4.12.10 would not be possible if a
1167 **PATH** change retrieved an alien version of `command`. (An alter-
1168 native would have been to implement `getconf` as a built-in, but
1169 it was felt that it carried too many changing configuration strings
1170 to require in the shell.)

1171 `kill` Since common extensions to `kill` (including the planned User
1172 Portability Extension) provide optional job control functionality
1173 using shell notation (`%1`, `%2`, etc.), some implementations would

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1174 find it extremely difficult to provide this outside the shell.

1175 true, false

1176 These are in the table as a courtesy to programmers who wish to

1177 use the “while true” shell construct without protecting true

1178 from **PATH** searches. (It is acknowledged that “while :” also

1179 works, but the idiom with true is historically pervasive.)

1180 All utilities, including those in the table, are accessible via the functions in 7.1.1

1181 or 7.1.2 [such as *system()* or *popen()*]. There are situations where the return func-

1182 tionality of *system()* and *popen()* is not desirable. Applications that require the

1183 exit status of the invoked utility will not be able to use *system()* or *popen()*, since

1184 the exit status returned is that of the command language interpreter rather than

1185 that of the invoked utility. The alternative for such applications is the use of the

1186 *exec* family. (The text concerning conformance to POSIX.1 {8} was included

1187 because where *exec* is not provided in the underlying system, there is no way to

1188 require that utilities be *exec*-able).

2.4 Character Set

Table 2-3 – Character Set and Symbolic Names

Symbolic Name	Glyph	Symbolic Name	Glyph	Symbolic Name	Glyph
<NUL>	<colon>	:	<circumflex>	^	
<alert>	<semicolon>	;	<circumflex-accent>	^	
<backspace>	<less-than-sign>	<	<underscore>	_	
<tab>	<equals-sign>	=	<low-line>	`	
<newline>	<greater-than-sign>	>	<grave-accent>	`	
<vertical-tab>	<question-mark>	?	<a>	a	
<form-feed>	<commercial-at>	@		b	
<carriage-return>	<A>	A	<c>	c	
<space>		B	<d>	d	
<exclamation-mark>	! <C>	C	<e>	e	
<quotation-mark>	" <D>	D	<f>	f	
<number-sign>	# <E>	E	<g>	g	
<dollar-sign>	\$ <F>	F	<h>	h	
<percent-sign>	% <G>	G	<i>	i	
<ampersand>	& <H>	H	<j>	j	
<apostrophe>	' <I>	I	<k>	k	
<left-parenthesis>	(<J>	J	<l>	l	
<right-parenthesis>) <K>	K	<m>	m	
<asterisk>	* <L>	L	<n>	n	
<plus-sign>	+ <M>	M	<o>	o	
<comma>	, <N>	N	<p>	p	
<hyphen>	- <O>	O	<q>	q	
<hyphen-minus>	- <P>	P	<r>	r	
<period>	. <Q>	Q	<s>	s	
<full-stop>	. <R>	R	<t>	t	
<slash>	/ <S>	S	<u>	u	
<solidus>	/ <T>	T	<v>	v	
<zero>	0 <U>	U	<w>	w	
<one>	1 <V>	V	<x>	x	
<two>	2 <W>	W	<y>	y	
<three>	3 <X>	X	<z>	z	
<four>	4 <Y>	Y	<left-brace>	{	
<five>	5 <Z>	Z	<left-curly-bracket>	{	
<six>	6 <left-square-bracket>	[<vertical-line>		
<seven>	7 <backslash>	\	<right-brace>	}	
<eight>	8 <reverse-solidus>	\	<right-curly-bracket>	}	
<nine>	9 <right-square-bracket>]	<tilde>	~	

Conforming implementations shall support one or more coded character sets. Each supported coded character set shall include the *portable character set* specified in Table 2-3. The table 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 names are chosen to correspond closely with character names defined in other international standards. The table contains more than one symbolic character name for characters whose traditional

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

name differs from the chosen name.

This standard places only the following requirements on the encoded values of the characters in the portable character set:

- (1) If the encoded values associated with each member of the portable character set are not invariant across all locales supported by the implementation, the results achieved by an application accessing those locales are unspecified.
- (2) The encoded values associated with the digits '0' to '9' shall be such that the value of each character after '0' shall be one greater than the value of the previous character.
- (3) A null character, NUL, which has all bits set to zero, shall be in the set of characters.

Conforming implementations shall support certain character and character set attributes, as defined in 2.5.1.

2.4.1 Character Set Description File

Implementations shall provide a character set description file for at least one coded character set supported by the implementation. These files are referred to elsewhere in this standard as *charmap* files. It is implementation defined whether or not users or applications can provide additional character set description files. If such a capability is supported, the system documentation shall describe the rules for the creation of such files.

Each character set description file shall define characteristics for the coded character set and the encoding for the characters specified in Table 2-3, and may define encoding for additional characters supported by the implementation. Other information about the coded character set may also be in the file. Coded character set character values shall be defined using symbolic character names followed by character encoding values.

Each symbolic name specified in Table 2-3 shall be included in the file and shall be mapped to a unique encoding value (except for those symbolic names that are shown with identical glyphs). If the control characters commonly associated with the symbolic names in Table 2-4 are supported by the implementation, the symbolic names and their corresponding encoding values shall be included in the file. Some of the values associated with the symbolic names in this table also may be contained in Table 2-3.

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 of more <blank>s, followed by the value to be assigned to the symbol.

<p><code_set_name></p>	<p>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</p>	<p>1</p>
------------------------------	---	----------

Table 2-4 – Control Character Set

1280							
1281							
1282	<ACK>	<DC2>	<ENQ>	<FS>	<IS4>	<SOH>	1
1283	<BEL>	<DC3>	<EOT>	<GS>	<LF>	<STX>	1
1284	<BS>	<DC4>	<ESC>	<HT>	<NAK>	<SUB>	1
1285	<CAN>		<ETB>	<IS1>	<RS>	<SYN>	1
1286	<CR>	<DLE>	<ETX>	<IS2>	<SI>	<US>	1
1287	<DC1>		<FF>	<IS3>	<SO>	<VT>	1
1288							

1289 visible glyphs defined in Table 2-3. 1

1290 <mb_cur_max> The maximum number of bytes in a multibyte character.
1291 This shall default to 1.

1292 <mb_cur_min> An unsigned positive integer value that shall define the
1293 minimum number of bytes in a character for the encoded
1294 character set. The value shall be less than or equal to
1295 mb_cur_max. If not specified, the minimum number
1296 shall be equal to mb_cur_max.

1297 <escape_char> The escape character used to indicate that the characters
1298 following shall be interpreted in a special way, as defined
1299 later in this subclause. This shall default to backslash
1300 (\), which is the character glyph used in all the following
1301 text and examples, unless otherwise noted.

1302 <comment_char> The character, that when placed in column 1 of a char-
1303 map line, is used to indicate that the line shall be
1304 ignored. The default character shall be the number-sign
1305 (#).

1306 The character set mapping definitions shall be all the lines immediately following
1307 an identifier line containing the string CHARMAP starting in column 1, and preced-
1308 ing a trailer line containing the string END CHARMAP starting in column 1. Empty
1309 lines and lines containing a comment_char in the first column shall be ignored.
1310 Each noncomment line of the character set mapping definition (i.e., between the
1311 CHARMAP and END CHARMAP lines of the file) shall be in either of two forms:

1312 "%s %s %s\n", <symbolic-name>, <encoding>, <comments>

1313 or

1314 "%s...%s %s %s\n", <symbolic-name>, <symbolic-name>, <encoding>,
1315 <comments>

1316 In the first format, the line in the character set mapping definition defines a sin-
1317 gle symbolic name and a corresponding encoding. A symbolic name is one or more
1318 characters from the set shown with visible glyphs in Table 2-3, enclosed between
1319 angle brackets. A character following an escape character shall be interpreted as
1320 itself; for example, the sequence "<\\>" represents the symbolic name ">"

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

enclosed between angle brackets.

In the second format, the line in the character set mapping definition defines a range of one or more symbolic names. In this form, the symbolic names shall consist of zero or more nonnumeric characters from the set shown with visible glyphs in Table 2-3, followed by an integer formed by one or more decimal 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 2-3, and shall define the coded character value that corresponds with the character glyph indicated in the table, or the coded character value that corresponds with the control character symbolic name. If the control characters commonly associated with the symbolic names in Table 2-4 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 shall be expressed as one (for single-byte character values) or more concatenated decimal, octal, or hexadecimal constants in the following formats:

`"%cd%d", <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 shall represent an 8-bit byte. Implementations supporting other byte sizes may allow constants to represent values larger than those that can be represented in 8-bit bytes, and to allow additional digits in constants. When constants are concatenated for multibyte character values, they shall be of the same type, and interpreted in byte order from left to right. The manner in which constants are represented in the character is implementation defined. Omitting bytes from a multibyte character definition produces undefined results.

In lines defining ranges of symbolic names, the encoded value is 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. For example, the line

```

1364      <j0101>...<j0104>      \d129\d254
1365  shall be interpreted as
1366      <j0101>                  \d129\d254
1367      <j0102>                  \d129\d255
1368      <j0103>                  \d130\d0
1369      <j0104>                  \d130\d1

```

1370 The comment is optional.

1371 For the interpretation of the dollar-sign and the number-sign, see 2.2.2.37 and
 1372 2.2.2.93.

1373 **2.4.2 Character Set Rationale.** *(This subclause is not a part of P1003.2)*

1374 The portable character set is listed in full so there is no dependency on the
 1375 ISO/IEC 646 {1} (or historically ASCII) encoded character set, although the set is
 1376 identical to the characters defined in the International Reference Version of
 1377 ISO/IEC 646 {1}.

1378 This standard poses no requirement that multiple character sets or code sets be
 1379 supported, leaving this as a marketing differentiation for implementors.
 1380 Although multiple *charmap* files are supported, it is the responsibility of the
 1381 implementation to provide the file(s); if only one is provided, only that one will be
 1382 accessible using the `localedef` utility's `-f` option (although in the case of just
 1383 one file on the system, `-f` is not useful).

1384 The statement about invariance in code sets for the portable character set is
 1385 worded as it is to avoid precluding implementations where multiple incompatible
 1386 code sets are available (say, ASCII and EBCDIC). The standard utilities cannot be
 1387 expected to produce predictable results if they access portable characters that
 1388 vary on the same implementation.

1389 The character set description file provides:

- 1390 — the capability to describe character set attributes (such as collation order or
 1391 character classes) independent of character set encoding, and using only
 1392 the characters in the portable character set. This makes it possible to
 1393 create “generic” `localedef` source files for all code sets that share the
 1394 portable character set (such as the ISO 8859 family or IBM Extended ASCII).
- 1395 — standardized symbolic names for all characters in the portable character
 1396 set, making it possible to refer to any such character regardless of encod-
 1397 ing.

1398 Implementations are free to describe more than one code set in a character set
 1399 description file, as long as only one encoding exists for the characters in Table 2-3.
 1400 For example, if an implementation defines ISO 8859-1 {5} as the primary code set,
 1401 and ISO 8859-2 {6} as an alternate set, with each character from the alternate
 1402 code set preceded in data by a shift code, a character set description file could con-
 1403 tain a complete description of the primary set and those characters from the
 1404 secondary that are not identical, the encoding of the latter including the shift

code.

Implementations are free to choose their own symbolic names, as long as the names identified by this standard are also defined; this provides support for already existing “character names.”

The names selected for the members of the portable character set follow the ISO 8859 {5} and the ISO/IEC 10646 {B11} standards. However, several commonly used UNIX system names occur as synonyms in the list:

- The traditional UNIX system names are used for control characters.
- The word “slash” is in addition to “solidus.”
- The word “backslash” is in addition to “reverse-solidus.”
- The word “hyphen” in addition to “hyphen-minus.”
- The word “period” in addition to “full-stop.”
- For the digits, the word “digit” is eliminated.
- For letters, the words “Latin Capital Letter” and “Latin Small Letter” are eliminated.
- The words “left-brace” and “right-brace” in addition to “left-curly-bracket” and “right-curly-bracket.”
- The names of the digits are preferred over the numbers, to avoid possible confusion between “0” and “O”, and between “1” and “l” (one and the letter ell).

The names for the control characters in Table 2-4 were taken from ISO 4873 {4}.

The charmap file was introduced to resolve problems with the portability of, especially, `localedef` sources. This standard assumes that the portable character set is constant across all locales, but does not prohibit implementations from supporting two incompatible codings, such as both ASCII and EBCDIC. Such “dual-support” implementations should have all charmaps and `localedef` sources encoded using one portable character set, in effect “cross-compiling” for the other environment. Naturally, charmaps (and `localedef` sources) are only portable without transformation between systems using the same encodings for the portable character set. They can, however, be transformed between two sets using only a subset of the actual characters (the portable set). However, the particular coded character set used for an application or an implementation does not necessarily imply different characteristics or collation: on the contrary, these attributes should in many cases be identical, regardless of code set. The charmap provides the capability to define a common locale definition for multiple code sets (the same `localedef` source can be used for code sets with different extended characters; the ability in the charmap to define “empty” names allows for characters missing in certain code sets).

In addition, several implementors have expressed an interest in using the charmap concept to provide the information required for support of multiple character sets. Examples of such information is encoding mechanism, string parsing rules,

1446 default font information, etc. Such extensions are not described here.

1447 The `<escape_char>` declaration was added at the request of the international
 1448 community to ease the creation of portable *charmap* files on terminals not imple-
 1449 menting the default backslash escape. (This approach was adopted because this
 1450 is a new interface invented by POSIX.2. Historical interfaces, such as the shell
 1451 command language and *awk*, have not been modified to accommodate this type of
 1452 terminal.) The `<comment_char>` declaration was added at the request of the
 1453 international community to eliminate the potential confusion between the number
 1454 sign and the pound sign.

1455 The octal number notation with no leading zero required was selected to match 1
 1456 those of *awk* and *tr* and is consistent with that used by *localedef*. To avoid 1
 1457 confusion between an octal constant and the backreferences used in *localedef* 1
 1458 source, the octal, hexadecimal, and decimal constants must contain at least two 1
 1459 digits. As single-digit constants are relatively rare, this should not impose any 1
 1460 significant hardship. Each of the constants includes “two or more” digits to 1
 1461 account for systems in which the byte size is larger than eight bits. For example, 1
 1462 a Unicode system that has defined 16-bit bytes may require six octal, four hexade- 1
 1463 cimal, and five decimal digits. 1

1464 The decimal notation is supported because some newer international standards
 1465 define character values in decimal, rather than in the old column/row notation.

1466 The *charmap* identifies the coded character sets supported by an implementation.
 1467 At least one *charmap* must be provided, but no implementation is required to pro-
 1468 vide more than one. Likewise, implementations can allow users to generate new
 1469 *charmaps* (for instance for a new version of the 8859 family of coded character
 1470 sets), but does not have to do so. If users are allowed to create new *charmaps*, the
 1471 system documentation must describe the rules that apply (for instance: “only
 1472 coded character sets that are supersets of ISO/IEC 646 {1} IRV, no multibyte char-
 1473 acters, etc.”)

2.5 Locale

A *locale* is the definition of the subset of a user's environment that depends on language and cultural conventions. It is made up from one or more categories. Each category is identified by its name and controls specific aspects of the behavior of components of the system. Category names correspond to the following environment variable names:

LC_CTYPE	Character classification and case conversion.
LC_COLLATE	Collation order.
LC_TIME	Date and time formats.
LC_NUMERIC	Numeric, nonmonetary formatting.
LC_MONETARY	Monetary formatting.
LC_MESSAGES	Formats of informative and diagnostic messages and interactive responses.

Conforming implementations shall provide the standard utilities and the interfaces in Annex B (if that option is supported) with the capability to modify their behavior based on the current locale, as defined in the Environment Variables subclause for each utility and interface.

Locales other than those supplied by the implementation can be created via the `localedef` utility (see 4.35), provided that the `{POSIX2_LOCALEDEF}` symbol is defined on the system; see 2.13.2. Otherwise, only the implementation-provided locale(s) can be used. The input to the utility is described in 2.5.2. The value that shall be used to specify a locale when using environment variables shall be the string specified as the *name* operand to the `localedef` utility when the locale was created. The strings "C" and "POSIX" are reserved as identifiers for the POSIX Locale (see 2.5.1.) When the value of a locale environment variable begins with a slash (/), it shall be interpreted as the pathname of the locale definition. If the value of the locale value does not begin with a slash, the mechanism used to locate the locale is implementation defined.

If different character sets are used by the locale categories, the results achieved by an application utilizing these categories is undefined. Likewise, if different code sets are used for the data being processed by interfaces whose behavior is dependent on the current locale, or the code set is different from the code set assumed when the locale was created, the result is also undefined.

1507 **2.5.0.1 Locale Rationale.** *(This subclause is not a part of P1003.2)*

1508 The description of locales is based on work performed in the UniForum Technical
1509 Committee Subcommittee on Internationalization. Wherever appropriate, key-
1510 words were taken from the C Standard [7] or the *X/Open Portability Guide* [B31].

1511 The value that shall be used to specify a locale when using environment variables
1512 is the name specified as the *name* operand to the `localedef` utility when the
1513 locale was created. This provides a verifiable method to create and invoke a
1514 locale.

1515 The “object” definitions need not be portable, as long as “source” definitions are.
1516 Strictly speaking, “source” definitions are portable only between implementations
1517 using the same character set(s). Such “source” definitions can, if they use sym-
1518 bolic names only, easily be ported between systems using different code sets as
1519 long as the characters in the portable character set (Table 2-3) have common
1520 values between the code sets; this is frequently the case in historical implementa-
1521 tions. Of course, this requires that the symbolic names used for characters out-
1522 side the portable character set are identical between character sets. The
1523 definition of symbolic names for characters is outside the scope of this standard,
1524 but is certainly within the scope of other standards organizations. When such
1525 names are standardized, future versions of POSIX.2 should require the use of
1526 these names.

1527 Applications can select the desired locale by invoking the `setlocale()` function (or
1528 equivalent) with the appropriate value. If the function is invoked with an empty
1529 string, the value of the corresponding environment variable is used. If the
1530 environment variable is unset or is set to the empty string, the implementation
1531 sets the appropriate environment as defined in 2.6.

1532 **2.5.1 POSIX Locale**

1533 Conforming implementations shall provide a *POSIX Locale*. The behavior of stan-
1534 dard utilities in the POSIX Locale shall be as if the locale was defined via the
1535 `localedef` utility with input data from Table 2-5, Table 2-7, Table 2-9, Table 2-
1536 10, Table 2-8, and Table 2-11, all in 2.5.2.

1537 The tables describe the characteristics and behavior of the POSIX Locale for data
1538 consisting entirely of characters from the portable character set in Table 2-3 and
1539 the control characters in Table 2-4. For characters other than those in the two
1540 tables, the behavior is unspecified.

1541 The POSIX Locale can be specified by assigning the appropriate environment vari-
1542 ables the values “C” or “POSIX”.

1543 Table 2-5 shows the definition for the LC_CTYPE category.

1544 Table 2-7 shows the definition for the LC_COLLATE category.

1545 Table 2-8 shows the definition for the LC_MONETARY category.

1546 Table 2-9 shows the definition for the LC_NUMERIC category.

1547 Table 2-10 shows the definition for the LC_TIME category.

1548 Table 2-11 shows the definition for the LC_MESSAGES category.

1549 **2.5.1.1 POSIX Locale Rationale.** *(This subclause is not a part of P1003.2)*

1550 The POSIX Locale is equal to the "C" locale, as specified in POSIX.1 {8}. To avoid
1551 being classified as a C-language function, the name has been changed to the
1552 *POSIX Locale*; the environment variable value can be either "POSIX", or, for his-
1553 torical reasons, "C".

1554 The POSIX definitions mirror the historical UNIX system behavior.

1555 The use of symbolic names for characters in the tables does not imply that the
1556 POSIX Locale must be described using symbolic character names, but merely that
1557 it may be advantageous to do so.

1558 Implementations must define a locale as the "default" locale, to be invoked when
1559 no environment variables are set, or set to the empty string. This default locale
1560 can be the POSIX Locale or any other, implementation-defined locale. Some
1561 implementations may provide facilities for local installation administrators to set
1562 the default locale, customizing it for each location. This standard does not require
1563 such a facility.

1

1564 **2.5.2 Locale Definition**

1565 The capability to specify additional locales to those provided by an implementa-
1566 tion is optional (see 2.13.2). If the option is not supported, only implementation-
1567 supplied locales are available. Such locales shall be documented using the format
1568 specified in this clause.

1569 Locales can be described with the file format presented in this subclause. The file
1570 format is that accepted by the `localedef` utility (see 4.35). For the purposes of
1571 this subclause, the file is referred to as the *locale definition file*, but no locales
1572 shall be affected by this file unless it is processed by `localedef` or some similar
1573 mechanism. Any requirements in this subclause imposed upon "the utility" shall
1574 apply to `localedef` or to any other similar utility used to install locale informa-
1575 tion using the locale definition file format described here.

1

1

1

1576 The locale definition file shall contain one or more locale category source
1577 definitions, and shall not contain more than one definition for the same locale
1578 category. If the file contains source definitions for more than one category,
1579 implementation-defined categories, if present, shall appear after the categories
1580 defined by this clause (2.5). A category source definition shall contain either the
1581 definition of a category or a `copy` directive. For a description of the `copy` direc-
1582 tive, see 4.35. In the event that some of the information for a locale category, as
1583 specified in this standard, is missing from the locale source definition, the
1584 behavior of that category, if it is referenced, is unspecified.

1585 A category source definition shall consist of a category header, a category body,
 1586 and a category trailer. A category header shall consist of the character string
 1587 naming of the category, beginning with the characters LC_. The category trailer
 1588 shall consist of the string END, followed by one or more <blank>s and the string 1
 1589 used in the corresponding category header.

1590 The category body shall consist of one or more lines of text. Each line shall con-
 1591 tain an identifier, optionally followed by one or more operands. Identifiers shall
 1592 be either keywords, identifying a particular locale element, or collating elements.
 1593 In addition to the keywords defined in this standard, the source can contain
 1594 implementation-defined keywords. Each keyword within a locale shall have a
 1595 unique name (i.e., two categories cannot have a commonly-named keyword); no
 1596 keyword shall start with the characters LC_. Identifiers shall be separated from
 1597 the operands by one or more <blank>s.

1598 Operands shall be characters, collating elements, or strings of characters. Strings 1
 1599 shall be enclosed in double-quotes. Literal double-quotes within strings shall be 1
 1600 preceded by the <escape character>, described below. When a keyword is followed 1
 1601 by more than one operand, the operands shall be separated by semicolons;
 1602 <blank>s shall be allowed before and/or after a semicolon.

1603 The first category header in the file can be preceded by a line modifying the com-
 1604 ment character. It shall have the following format, starting in column 1:

1605 "comment_char %c\n", <comment character>

1606 The comment character shall default to the number-sign (#). Blank lines and
 1607 lines containing the <comment char> in the first position shall be ignored.

1608 The first category header in the file can be preceded by a line modifying the
 1609 escape character to be used in the file. It shall have the following format, starting
 1610 in column 1:

1611 "escape_char %c\n", <escape character>

1612 The escape character shall default to backslash, which is the character used in all
 1613 examples shown in this standard.

1614 A line can be continued by placing an escape character as the last character on
 1615 the line; this continuation character shall be discarded from the input. Although 1
 1616 the implementation need not accept any one portion of a continued line with a 1
 1617 length exceeding {LINE_MAX} bytes, it shall place no limits on the accumulated 1
 1618 length of the continued line. Comment lines shall not be continued on a subse- 1
 1619 quent line using an escaped <newline>.

1620 Individual characters, characters in strings, and collating elements shall be 2
 1621 represented using symbolic names, as defined below. In addition, characters can 2
 1622 be represented using the characters themselves, or as octal, hexadecimal, or 2
 1623 decimal constants. When nonsymbolic notation is used, the resultant locale 2
 1624 definitions need not be portable between systems. The left angle bracket (<) is a 2
 1625 reserved symbol, denoting the start of a symbolic name; when used to represent 2
 1626 itself it shall be preceded by the escape character. The following rules apply to 2
 1627 character representation: 2

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

- (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 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` in 4.35 for a description of action resulting from errors and warnings). The specification of a symbolic name in a `collating-element` or `collating-symbol` clause that duplicates a symbolic name in the charmap file (if present) is an error. Use of the escape character or a right angle bracket within a symbolic name shall be invalid unless the character is preceded by the escape character.

Example: `<c>;<c-cedilla> " <M><a><y> "`

- (2) A character can be represented by the character itself, in which case the value of the character is implementation defined. 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

`, ; < > escape_char`

shall be escaped to be interpreted as the character itself.

Example: `c β "May"`

- (3) A character can be represented as an octal constant. An octal constant shall be specified as the escape character followed by two or more octal digits. Each constant shall represent a byte value. Multibyte characters can be represented by concatenated constants.

Example: `\143;\347;\143\150 "\115\141\171"`

- (4) A character can be represented as a hexadecimal constant. A hexadecimal constant shall be specified as the escape character followed by an `x` followed by two or more hexadecimal digits. Each constant shall represent a byte value. Multibyte characters can be represented by concatenated constants.

Example: `\x63;\xe7;\x63\x68 "\x4d\x61\x79"`

- (5) A character can be represented as a decimal constant. A decimal constant shall be specified as the escape character followed by a `d` followed by two or more decimal digits. Each constant shall represent a byte value. Multibyte values can be represented by concatenated constants.

Example: `\d99;\d231;\d99\d104 "\d77\d97\d121"`

Implementations may accept single-digit octal, decimal, or hexadecimal constants following the escape character. Only characters existing in the character set for which the locale definition is created shall be specified, whether using symbolic

names, the characters themselves, or octal, decimal, or hexadecimal constants. If a charmap file is present, only characters defined in the charmap can be specified using octal, decimal, or hexadecimal constants. Symbolic names not present in the charmap file can be specified and shall be ignored, as specified under item (1) above.

2.5.2.0.1 Locale Definition Rationale. *(This subclause is not a part of P1003.2)*

The decision to separate the file format from the `localedef` utility description was only partially editorial. Implementations may provide other interfaces than `localedef`. Requirements on “the utility,” mostly concerning error messages, are described in this way because they are meant to affect the other interfaces implementations may provide as well as `localedef`. (This is similar to the philosophy used by POSIX.1 {8} where the descriptions of the `tar` and `cpio` file formats impose requirements on any utilities processing them.)

The text about `{POSIX2_LOCALEDEF}` does not mean that internationalization is optional; only that the functionality of the `localedef` utility is. Regular expressions, for instance, must still be able to recognize e.g., character class expressions such as `[[:alpha:]]`.

A possible analogy is with an applications development environment: while all conforming implementations must be capable of executing applications, not all need to have the development environment installed. The assumption is that the capability to modify the behavior of utilities (and applications) via locale settings must be supported. If the `localedef` utility is not present, then the only choice is to select an existing (presumably implementation-documented) locale. An implementation could, for example, chose to support only the POSIX Locale, which would in effect limit the amount of changes from historical implementations quite drastically. The `localedef` utility is still required, but would always terminate with an exit code indicating that no locale could be created. Supported locales must be documented using the syntax defined in 2.5. (This ensures that users can accurately determine what capabilities are provided. If the implementation decides to provide additional capabilities to the ones in 2.5, that is already provided for.)

If the option is present (i.e., locales can be created), then the `localedef` utility must be capable of creating locales based on the syntax and rules defined in 2.5. This does not mean that the implementation cannot also provide alternate means for creating locales.

The octal, decimal, and hexadecimal notations are the same employed by the charmap facility (see 2.4.1). To avoid confusion between an octal constant and a backreference, the octal, hexadecimal, and decimal constants must contain at least two digits. As single-digit constants are relatively rare, this should not impose any significant hardship. Each of the constants includes “two or more” digits to account for systems in which the byte size is larger than eight bits. For example, a Unicode system that has defined 16-bit bytes may require six octal, four hexadecimal, and five decimal digits.

This standard is intended as an international (ISO/IEC) standard as well as an IEEE standard, and must therefore follow the ISO/IEC guidelines. One such rule is that characters outside the invariant part of ISO/IEC 646 {1} should not be used in portable specifications. The backslash character is not in the invariant part; the number-sign is, but with multiple representations: as a number-sign and as a pound sign. As far as general usage of these symbols, they are covered by the “grandfather clause,” but for newly defined interfaces, ISO has requested that POSIX provides alternate representations. Consequently, while the default escape character remains the backslash, and the default comment character is the number-sign, implementations are required to recognize alternative representations, identified in the applicable source file via the `escape_char` and `comment_char` keywords.

2.5.2.1 LC_CTYPE

The LC_CTYPE category shall define character classification, case conversion, and other character attributes. In addition, a series of characters can be represented by three adjacent periods representing an ellipsis symbol (“...”). The ellipsis specification shall be interpreted as meaning that all values between the values preceding and following it represent valid characters. The ellipsis specification only shall be valid within a single encoded character set. An ellipsis shall be interpreted as including in the list all characters with an encoded value higher than the encoded value of the character preceding the ellipsis and lower than the encoded value of the character following the ellipsis.

Example: `\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 to either include the referenced characters or to omit them; the implementation shall provide them if missing and accept them silently if present.

<code>copy</code>	Specify the name of an existing locale to be used as the source for the definition of this category. If this keyword is specified, no other keyword shall be specified.
<code>upper</code>	Define characters to be classified as uppercase letters. No character specified for the keywords <code>cntrl</code> , <code>digit</code> , <code>punct</code> , or <code>space</code> shall be specified. If this keyword is not specified, the uppercase letters A through Z, as defined in Table 2-3 (see 2.4.1), shall automatically belong to this class, with implementation-defined character values.
<code>lower</code>	Define characters to be classified as lowercase letters. No character specified for the keywords <code>cntrl</code> , <code>digit</code> , <code>punct</code> , or <code>space</code> shall be specified. If this keyword is not specified, the lowercase letters a through z, as defined in Table 2-3 (see 2.4.1), shall automatically belong to this class, with implementation-defined character values.

Table 2-5 – LC_CTYPE Category Definition in the POSIX Locale

```

1756
1757
1758 LC_CTYPE
1759 # The following is the POSIX Locale LC_CTYPE.
1760 # "alpha" is by default "upper" and "lower"
1761 # "alnum" is by definition "alpha" and "digit"
1762 # "print" is by default "alnum", "punct" and the <space> character
1763 # "graph" is by default "alnum" and "punct"
1764 #
1765 upper <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;<M>;\
1766      <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>
1767 #
1768 lower <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
1769      <n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>
1770 #
1771 digit <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;<eight>;<nine>
1772 #
1773 space <tab>;<newline>;<vertical-tab>;<form-feed>;<carriage-return>;<space>
1774 #
1775 cntrl <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\
1776      <form-feed>;<carriage-return>;\
1777      <NUL>;<SOH>;<STX>;<ETX>;<EOT>;<ENQ>;<ACK>;<SO>;\
1778      <SI>;<DLE>;<DC1>;<DC2>;<DC3>;<DC4>;<NAK>;<SYN>;\
1779      <ETB>;<CAN>;<EM>;<SUB>;<ESC>;<IS4>;<IS3>;<IS2>;\
1780      <IS1>;<DEL>
1781 #
1782 punct <exclamation-mark>;<quotation-mark>;<number-sign>;\
1783      <dollar-sign>;<percent-sign>;<ampersand>;<apostrophe>;\
1784      <left-parenthesis>;<right-parenthesis>;<asterisk>;\
1785      <plus-sign>;<comma>;<hyphen>;<period>;<slash>;\
1786      <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
1787      <greater-than-sign>;<question-mark>;<commercial-at>
1788      <left-square-bracket>;<backslash>;<right-square-bracket>;\
1789      <circumflex>;<underline>;<grave-accent>;\
1790      <left-curly-bracket>;<vertical-line>;<right-curly-bracket>;<tilde>
1791 #
1792 xdigit <zero>;<one>;<two>;<three>;<four>;<five>;<six>;<seven>;<eight>;\
1793      <nine>;<A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
1794 #
1795 blank <space>;<tab>
1796 #
1797 toupper (<a>,<A>);(<b>,<B>);(<c>,<C>);(<d>,<D>);(<e>,<E>);\
1798      (<f>,<F>);(<g>,<G>);(<h>,<H>);(<i>,<I>);(<j>,<J>);\
1799      (<k>,<K>);(<l>,<L>);(<m>,<M>);(<n>,<N>);(<o>,<O>);\
1800      (<p>,<P>);(<q>,<Q>);(<r>,<R>);(<s>,<S>);(<t>,<T>);\
1801      (<u>,<U>);(<v>,<V>);(<w>,<W>);(<x>,<X>);(<y>,<Y>);(<z>,<Z>)
1802 #
1803 tolower (<A>,<a>);(<B>,<b>);(<C>,<c>);(<D>,<d>);(<E>,<e>);\
1804      (<F>,<f>);(<G>,<g>);(<H>,<h>);(<I>,<i>);(<J>,<j>);\
1805      (<K>,<k>);(<L>,<l>);(<M>,<m>);(<N>,<n>);(<O>,<o>);\
1806      (<P>,<p>);(<Q>,<q>);(<R>,<r>);(<S>,<s>);(<T>,<t>);\
1807      (<U>,<u>);(<V>,<v>);(<W>,<w>);(<X>,<x>);(<Y>,<y>);(<Z>,<z>)
1808 END LC_CTYPE
1809

```

1810	alpha	Define characters to be classified as letters. No character	
1811		specified for the keywords <code>cntrl</code> , <code>digit</code> , <code>punct</code> , or <code>space</code> shall	
1812		be specified. In addition, characters classified as either upper	
1813		or lower shall automatically belong to this class.	
1814	digit	Define the characters to be classified as numeric digits. Only the	2
1815		digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 shall be specified, and in	2
1816		ascending sequence by numerical value. If this keyword is not	2
1817		specified, the digits 0 through 9, as defined in Table 2-3 (see	2
1818		2.4.1), shall automatically belong to this class, with	2
1819		implementation-defined character values.	2
1820	space	Define characters to be classified as white-space characters. No	
1821		character specified for the keywords <code>upper</code> , <code>lower</code> , <code>alpha</code> ,	
1822		<code>digit</code> , <code>graph</code> , or <code>xdigit</code> shall be specified. If this keyword is	2
1823		not specified, the characters <code><space></code> , <code><form-feed></code> , <code><new-</code>	2
1824		<code>line></code> , <code><carriage-return></code> , <code><tab></code> , and <code><vertical-tab></code> , as	2
1825		defined in Table 2-3 (see 2.4.1), shall automatically belong to	2
1826		this class, with implementation-defined character values. Any	2
1827		characters included in the class <code>blank</code> shall be automatically	1
1828		included.	1
1829	cntrl	Define characters to be classified as control characters. No char-	
1830		acter specified for the keywords <code>upper</code> , <code>lower</code> , <code>alpha</code> , <code>digit</code> ,	
1831		<code>punct</code> , <code>graph</code> , <code>print</code> , or <code>xdigit</code> shall be specified.	1
1832	punct	Define characters to be classified as punctuation characters. No	
1833		character specified for the keywords <code>upper</code> , <code>lower</code> , <code>alpha</code> ,	
1834		<code>digit</code> , <code>cntrl</code> , <code>xdigit</code> , or as the <code><space></code> character shall be	
1835		specified.	
1836	graph	Define characters to be classified as printable characters, not	
1837		including the <code><space></code> character. If this keyword is not	
1838		specified, characters specified for the keywords <code>upper</code> , <code>lower</code> ,	
1839		<code>alpha</code> , <code>digit</code> , <code>xdigit</code> , and <code>punct</code> shall belong to this character	
1840		class. No character specified for the keyword <code>cntrl</code> shall be	
1841		specified.	
1842	print	Define characters to be classified as printable characters, includ-	
1843		ing the <code><space></code> character. If this keyword is not provided,	
1844		characters specified for the keywords <code>upper</code> , <code>lower</code> , <code>alpha</code> ,	
1845		<code>digit</code> , <code>xdigit</code> , <code>punct</code> , and the <code><space></code> character shall belong	
1846		to this character class. No character specified for the keyword	
1847		<code>cntrl</code> shall be specified.	
1848	xdigit	Define the characters to be classified as hexadecimal digits.	2
1849		Only the characters defined for the class <code>digit</code> shall be	2
1850		specified, in ascending sequence by numerical value, followed by	2
1851		one or more sets of six characters representing the hexadecimal	2
1852		digits 10 through 15, with each set in ascending order (for exam-	2
1853		ple A, B, C, D, E, F, a, b, c, d, e, f). If this keyword is not	2
1854		specified, the digits 0 through 9, the uppercase letters A through	2

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1855		F, and the lowercase letters a through f, as defined in Table 2-3	2
1856		(see 2.4.1), shall automatically belong to this class, with	2
1857		implementation-defined character values.	2
1858	blank	Define characters to be classified as <blank> characters. If this	
1859		keyword is unspecified, the characters <space> and <tab> shall	
1860		belong to this character class.	
1861	toupper	Define the mapping of lowercase letters to uppercase letters.	
1862		The operand shall consist of character pairs, separated by semi-	
1863		colons. The characters in each character pair shall be separated	
1864		by a comma and the pair enclosed by parentheses. The first	
1865		character in each pair shall be the lowercase letter, the second	
1866		the corresponding uppercase letter. Only characters specified for	
1867		the keywords lower and upper shall be specified. If this key-	2
1868		word is not specified, the lowercase letters a through z, and	2
1869		their corresponding uppercase letters A through Z, as defined in	2
1870		Table 2-3 (see 2.4.1), shall automatically be included, with	2
1871		implementation-defined character values.	2
1872	tolower	Define the mapping of uppercase letters to lowercase letters.	
1873		The operand shall consist of character pairs, separated by semi-	
1874		colons. The characters in each character pair are separated by a	
1875		comma and the pair enclosed by parentheses. The first charac-	
1876		ter in each pair shall be the uppercase letter, the second the	
1877		corresponding lowercase letter. Only characters specified for the	
1878		keywords lower and upper shall be specified.	
1879		The tolower keyword is optional. If specified, the uppercase	
1880		letters A through Z, as defined in Table 2-3, and their	
1881		corresponding lowercase letter, shall be specified. If this key-	
1882		word is not specified, the mapping shall be the reverse mapping	
1883		of the one specified for toupper.	
1884	Table 2-6 shows the allowed character class combinations.		

Table 2-6 – Valid Character Class Combinations

In Class	Can Also Belong To										
	upper	lower	alpha	digit	space	cntrl	punct	graph	print	xdigit	blank
upper	-	-	M	X	X	X	X	D	D	-	X
lower	-	-	M	X	X	X	X	D	D	-	X
alpha	-	-	-	X	X	X	X	D	D	-	X
digit	X	X	X	-	X	X	X	D	D	-	X
space	X	X	X	X	-	-	*	*	*	X	-
cntrl	X	X	X	X	-	-	X	X	X	X	-
punct	X	X	X	X	-	X	-	D	D	X	-
graph	-	-	-	-	-	X	-	-	-	-	-
print	-	-	-	-	-	X	-	-	-	-	-
xdigit	-	-	-	-	X	X	X	D	D	-	X
blank	X	X	X	X	M	-	*	*	*	X	-

NOTES:

(1) Explanation of codes:

M	Always
D	Default; belongs to class if not specified
-	Permitted
X	Mutually exclusive
*	See note (2)

(2) The <space> character, which is part of the space and blank classes, cannot belong to punct or graph, but automatically shall belong to the print class. Other space or blank characters can be classified as punct, graph, and/or print.

2.5.2.1.1 LC_CTYPE Rationale. *(This subclause is not a part of P1003.2)*

The LC_CTYPE category primarily is used to define the encoding-independent aspects of a character set, such as character classification. In addition, certain encoding-dependent characteristics are also defined for an application via the LC_CTYPE category. POSIX.2 does not mandate that the encoding used in the locale is the same as the one used by the application, because an implementation may decide that it is advantageous to define locales in a system-wide encoding rather than having multiple, logically identical locales in different encodings, and to convert from the application encoding to the system-wide encoding on usage. Other implementations could require encoding-dependent locales.

In either case, the LC_CTYPE attributes that are directly dependent on the encoding, such as `mb_cur_max` and the display width of characters, are not user-specifiable in a locale source, and are consequently not defined as keywords.

As the LC_CTYPE character classes are based on the C Standard {7} character-class definition, the category does not support multicharacter elements. For

instance, the German character <sharp-s> is traditionally classified as a lower-case letter. There is no corresponding uppercase letter; in proper capitalization of German text the <sharp-s> will be replaced by SS; i.e., by two characters. This kind of conversion is outside the scope of the `toupper` and `tolower` keywords.

Where POSIX.2 specifies that only certain characters can be specified, as for the keywords `digit` and `xdigit`, the specified characters must be from the portable character set, as shown. As an example, only the Arabic digits 0 through 9 are acceptable as digits.

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 (i.e., encoding) differs from the implementation default values.

The definition of character class `digit` requires that only ten characters—the ones defining digits—can be specified; alternate digits (e.g., Hindi or Kanji) cannot be specified here. However, the encoding may vary if an implementation supports more than one encoding.

The definition of character class `xdigit` requires that the characters included in character class `digit` are included here also, and allows for different symbols for the hexadecimal digits 10 through 15.

2.5.2.2 LC_COLLATE

A collation sequence definition shall define the relative order between collating elements (characters and multicharacter collating elements) in the locale. This order is expressed in terms of collation values; i.e., by assigning each element one or more collation values (also known as collation weights). This does not imply that implementations shall assign such values, but that ordering of strings using the resultant collation definition in the locale shall behave as if such assignment is done and used in the collation process. The collation sequence definition shall be used by regular expressions, pattern matching, and sorting. The following capabilities are provided:

- (1) **Multicharacter collating elements.** Specification of multicharacter collating elements (i.e., 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}`) 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.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

- 1968 (5) **Many-to-Many substitution.** A string of one or more characters is sub-
 1969 stituted by another string (or an empty string, i.e., the character or char-
 1970 acters shall be ignored for collation purposes).
- 1971 (6) **Equivalence class definition.** Two or more collating elements have
 1972 the same collation value (primary weight).
- 1973 (7) **Ordering by weights.** When two strings are compared to determine 2
 1974 their relative order, the two strings are first broken up into a series of 2
 1975 collating elements, and each successive pair of elements are compared 2
 1976 according to the relative primary weights for the elements. If equal, and 2
 1977 more than one weight has been assigned, then the pairs of collating ele- 2
 1978 ments are recompared according to the relative subsequent weights, until 2
 1979 either a pair of collating elements compare unequal or the weights are 2
 1980 exhausted. 2
- 1981 The following keywords shall be recognized in a collation sequence definition.
 1982 They are described in detail in the following subclauses.
- 1983 `copy` Specify the name of an existing locale to be used as the
 1984 source for the definition of this category. If this key-
 1985 word is specified, no other keyword shall be specified.
- 1986 `collating-element` Define a collating-element symbol representing a mul- 1
 1987 ticharacter collating element. This keyword is 1
 1988 optional.
- 1989 `collating-symbol` Define a collating symbol for use in collation order 1
 1990 statements. This keyword is optional. 1
- 1991 `order_start` Define collation rules. This statement is followed by
 1992 one or more collation order statements, assigning char-
 1993 acter collation values and collation weights to collating
 1994 elements.
- 1995 `order_end` Specify the end of the collation-order statements. 1
- 1996 **2.5.2.2.1 collating-element Keyword**
- 1997 In addition to the collating elements in the character set, the `collating-`
 1998 `element` keyword shall be used to define multicharacter collating elements. The
 1999 syntax is
- 2000 `"collating-element %s from %s\n", <collating-symbol>, <string>`
- 2001 The `<collating-symbol>` operand shall be a symbolic name, enclosed between 1
 2002 angle brackets (< and >), and shall not duplicate any symbolic name in the
 2003 current charmap file (if any), or any other symbolic name defined in this collation
 2004 definition. The string operand shall be a string of two or more characters that
 2005 shall collate as an entity. A `<collating-element>` defined via this keyword is only 1
 2006 recognized with the LC_COLLATE category.

Table 2-7 – LC_COLLATE Category Definition in the POSIX Locale

```

2007 LC_COLLATE
2008
2009 # This is the POSIX Locale definition for the LC_COLLATE category.
2010 # The order is the same as in the ASCII code set.
2011 order_start forward
2012 <NUL>
2013 <SOH>
2014 <STX>
2015 <ETX>
2016 <EOT>
2017 <ENQ>
2018 <ACK>
2019 <alert>
2020 <backspace>
2021 <tab>
2022 <newline>
2023 <vertical-tab>
2024 <form-feed>
2025 <carriage-return>
2026 <SO>
2027 <SI>
2028 <DLE>
2029 <DC1>
2030 <DC2>
2031 <DC3>
2032 <DC4>
2033 <NAK>
2034 <SYN>
2035 <ETB>
2036 <CAN>
2037 <EM>
2038 <SUB>
2039 <ESC>
2040 <IS4>
2041 <IS3>
2042 <IS2>
2043 <IS1>
2044 <space>
2045 <exclamation-mark>
2046 <quotation-mark>
2047 <number-sign>
2048 <dollar-sign>
2049 <percent-sign>
2050 <ampersand>
2051 <apostrophe>
2052 <left-parenthesis>
2053 <right-parenthesis>
2054 <asterisk>
2055
2056

```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 2-7 – LC_COLLATE Category Definition in the POSIX Locale (*continued*)

2057	
2058	
2059	<plus-sign>
2060	<comma>
2061	<hyphen>
2062	<period>
2063	<slash>
2064	<zero>
2065	<one>
2066	<two>
2067	<three>
2068	<four>
2069	<five>
2070	<six>
2071	<seven>
2072	<eight>
2073	<nine>
2074	<colon>
2075	<semicolon>
2076	<less-than-sign>
2077	<equals-sign>
2078	<greater-than-sign>
2079	<question-mark>
2080	<commercial-at>
2081	<A>
2082	
2083	<C>
2084	<D>
2085	<E>
2086	<F>
2087	<G>
2088	<H>
2089	<I>
2090	<J>
2091	<K>
2092	<L>
2093	<M>
2094	<N>
2095	<O>
2096	<P>
2097	<Q>
2098	<R>
2099	<S>
2100	<T>
2101	<U>
2102	<V>
2103	<W>
2104	<X>
2105	<Y>
2106	<Z>
2107	

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

Table 2-7 – LC_COLLATE Category Definition in the POSIX Locale (*concluded*)

2108	
2109	
2110	<left-square-bracket>
2111	<backslash>
2112	<right-square-bracket>
2113	<circumflex>
2114	<underline>
2115	<grave-accent>
2116	<a>
2117	
2118	<c>
2119	<d>
2120	<e>
2121	<f>
2122	<g>
2123	<h>
2124	<i>
2125	<j>
2126	<k>
2127	<l>
2128	<m>
2129	<n>
2130	<o>
2131	<p>
2132	<q>
2133	<r>
2134	<s>
2135	<t>
2136	<u>
2137	<v>
2138	<w>
2139	<x>
2140	<y>
2141	<z>
2142	<left-curly-bracket>
2143	<vertical-line>
2144	<right-curly-bracket>
2145	<tilde>
2146	
2147	order_end
2148	#
2149	END LC_COLLATE
2150	
2151	<i>Example:</i>
2152	collating-element <ch> from <c><h>
2153	collating-element <e-acute> from <acute><e>
2154	collating-element <ll> from ll

2.5.2.2.2 collating-symbol Keyword

This keyword shall be used to define symbols for use in collation sequence statements; i.e., between the `order_start` and the `order_end` keywords. The syntax is

```
"collating-symbol %s\n", <collating-symbol>
```

The *<collating-symbol>* 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. A *<collating-symbol>* defined via this keyword is only recognized with the LC_COLLATE category.

Example:

```
collating-symbol <UPPER_CASE>
collating-symbol <HIGH>
```

2.5.2.2.3 order_start Keyword

The `order_start` keyword shall precede collation order entries and also defines the number of weights for this collation sequence definition and other collation rules.

The syntax of the `order_start` keyword is:

```
"order_start %s;%s;...;%s\n", <sort-rules>, <sort-rules> ...
```

The operands to the `order_start` keyword are optional. If present, the operands define rules to be applied when strings are compared. The number of operands define how many weights each element is assigned; if no operands are present, one forward operand is assumed. If present, the first operand defines rules to be applied when comparing strings using the first (primary) weight; the second when comparing strings using the second weight, and so on. Operands shall be separated by semicolons (;). Each operand shall consist of one or more collation directives, separated by commas (,). If the number or operands exceeds the {COLL_WEIGHTS_MAX} limit, the utility shall issue a warning message. The following directives shall be supported:

forward	Specifies that comparison operations for the weight level shall proceed from start of string towards the end of string.
---------	---

backward	Specifies that comparison operations for the weight level shall proceed from end of string towards the beginning of string.
----------	---

position	Specifies that comparison operations for the weight level will consider the relative position of non-IGNORED elements in the strings. The string containing a non-
----------	--

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2193	IGNORED element after the fewest IGNORED collating ele-	2
2194	ments from the start of the compare shall collate first. If	2
2195	both strings contain a non-IGNORED character in the same	2
2196	relative position, the collating values assigned to the ele-	2
2197	ments shall determine the ordering. In case of equality,	2
2198	subsequent non-IGNORED characters shall be considered in	2
2199	the same manner.	2
2200	The directives forward and backward are mutually exclusive.	
2201	<i>Example:</i>	
2202	order_start forward;backward	2
2203	If no operands are specified, a single forward operand shall be assumed.	1
2204	2.5.2.2.4 Collation Order	
2205	The order_start keyword shall be followed by collating element entries. The	
2206	syntax for the collating element entries is	
2207	"%s %s;%s;...;%s\n", <collating-element>, <weight>, <weight>, ...	
2208	Each <i>collating-element</i> shall consist of either a character (in any of the forms	1
2209	defined in 2.5.2), a <collating-element>, a <collating-symbol>, an ellipsis, or the	1
2210	special symbol UNDEFINED. The order in which collating elements are specified	1
2211	determines the character collation sequence, such that each collating element	1
2212	shall compare less than the elements following it. The NUL character shall com-	1
2213	pare lower than any other character.	1
2214	A <collating-element> shall be used to specify multicharacter collating elements,	1
2215	and indicates that the character sequence specified via the <collating-element> is	1
2216	to be collated as a unit and in the relative order specified by its place.	1
2217	A <collating-symbol> shall be used to define a position in the relative order for use	1
2218	in weights.	1
2219	The ellipsis symbol ("...") specifies that a sequence of characters shall collate	1
2220	according to their encoded character values. It shall be interpreted as indicating	1
2221	that all characters with a coded character set value higher than the value of the	
2222	character in the preceding line, and lower than the coded character set value for	
2223	the character in the following line, in the current coded character set, shall be	
2224	placed in the character collation order between the previous and the following	
2225	character in ascending order according to their coded character set values. An	
2226	initial ellipsis shall be interpreted as if the preceding line specified the NUL char-	
2227	acter, and a trailing ellipsis as if the following line specified the highest coded	
2228	character set value in the current coded character set. An ellipsis shall be treated	
2229	as invalid if the preceding or following lines do not specify characters in the	
2230	current coded character set. The use of the ellipsis symbol ties the definition to a	1
2231	specific coded character set and may preclude the definition from being portable	1
2232	between implementations.	1
2233	The symbol UNDEFINED shall be interpreted as including all coded character set	
2234	values not specified explicitly or via the ellipsis symbol. Such characters shall be	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

inserted in the character collation order at the point indicated by the symbol, and in ascending order according to their coded character set values. If no UNDEFINED symbol is specified, and the current coded character set contains characters not specified in this clause, the utility shall issue a warning message and place such characters at the end of the character collation order.

The optional operands for each collation-element shall be used to define the primary, secondary, or subsequent weights for the collating element. The first operand specifies the relative primary weight, the second the relative secondary weight, and so on. Two or more collation-elements can be assigned the same weight; they belong to the same *equivalence class* if they have the same primary weight. Collation shall behave as if, for each weight level, IGNORED elements are removed. Then each successive pair of elements shall be compared according to the relative weights for the elements. If the two strings compare equal, the process shall be repeated for the next weight level, up to the limit {COLL_WEIGHTS_MAX}.

Weights shall be expressed as characters (in any of the forms specified in 2.5.2), *<collating-symbol>*s, *<collating-element>*s, an ellipsis, or the special symbol IGNORE. A single character, a *<collating-symbol>*, or a *<collating-element>* shall represent the relative order in the character collating sequence of the character or symbol, rather than the character or characters themselves.

One-to-many mapping is indicated by specifying two or more concatenated characters or symbolic names. Thus, if the character “<eszet>” is given the string <s><s> as a weight, comparisons shall be performed as if all occurrences of the character <eszet> are replaced by <s><s>. If it is desirable to define <eszet> and <s><s> as an equivalence class, then a collating-element must be defined for the string “ss”, as in the example below.

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 (i.e., 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. Secondary and subsequent weights have unique values. 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; i.e., as if the string did not contain the collating element. In regular expressions and pattern matching, all characters that are IGNORED 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>;<a>
```

is equal to

2279 <a>

2280 An ellipsis can be used as an operand if the collating-element was an ellipsis, and
2281 shall be interpreted as the value of each character defined by the ellipsis.

2282 The collation order as defined in this clause defines the interpretation of bracket 1
2283 expressions in regular expressions (see 2.8.3.2). 1

2284 *Example:*

2285	order_start	forward;backward
2286	UNDEFINED	IGNORE;IGNORE
2287	<LOW>	
2288	<space>	<LOW>;<space>
2289	...	<LOW>;...
2290	<a>	<a>;<a>
2291	<a-acute>	<a>;<a-acute>
2292	<a-grave>	<a>;<a-grave>
2293	<A>	<a>;<A>
2294	<A-acute>	<a>;<A-acute>
2295	<A-grave>	<a>;<A-grave>
2296	<ch>	<ch>;<ch>
2297	<Ch>	<ch>;<Ch>
2298	<s>	<s>;<s>

```

2299         <esztet>          <s><s>;<esztet><esztet>
2300         ...                <HIGH>;i...
2301         <HIGH>
2302         order_end

```

2303 This example is interpreted as follows:

- (1) The UNDEFINED means that all characters not specified in this definition (explicitly or via the ellipsis) shall be ignored for collation purposes; for regular expression purposes they are ordered first.
- (2) All characters between <space> and <a> shall have the same primary equivalence class and individual secondary weights based on their ordinal encoded values.
- (3) All characters based on the upper- or lowercase character a belong to the same primary equivalence class.
- (4) The multicharacter collating element <c><h> is represented by the collating symbol <ch> and belongs to the same primary equivalence class as the multicharacter collating element <C><H>.
- (5) Note that it is not possible to use the collating element <ss> as a weight and expect it to be expanded to the string “ss”. When used as a weight, any collating-element represents the relative order assigned to it in the character collation sequence, not the string from which it was derived (compare with <ch>).

2.5.2.2.5 `order_end` Keyword

The collating order entries shall be terminated with an `order_end` keyword.

2.5.2.2.6 `LC_COLLATE` Rationale. *(This subclause is not a part of P1003.2)*

The `LC_COLLATE` category governs the collation order in the locale, and thus the processing of the C Standard {7} `strxfrm()` and `strcoll()` functions, as well as a number of POSIX.2 utilities.

The rules governing collation depends to some extent on the use. At least five different levels of increasingly complex collation rules can be distinguished:

- (1) Byte/machine code order. This is the historical collation order in the UNIX system and many proprietary operating systems. Collation is here done character by character, without any regard to context. The primary virtue is that it usually is quite fast, and also completely deterministic; it works well when the native machine collation sequence matches the user expectations.
- (2) Character order. On this level, collation is also done character by character, without regard to context. The order between characters is, however, not determined by the code values, but on the user's expectations of the "correct" order between characters. In addition, such a (simple) collation order can specify that certain characters collate equal (e.g., upper- and lowercase letters).
- (3) String ordering. On this level, entire strings are compared based on relatively straightforward rules. At this level, several "passes" may be required to determine the order between two strings. Characters may be ignored in some passes, but not in others; the strings may be compared in different directions; and simple string substitutions may be made before strings are compared. This level is best described as "dictionary" ordering; it is based on the spelling, not the pronunciation, or meaning, of the words.
- (4) Text search ordering. This is a further refinement of the previous level, best described as "telephone book ordering"; some common homonyms (words spelled differently but with same pronunciation) are collated together; numbers are collated as if spelled with words, and so on.
- (5) Semantic level ordering. Words and strings are collated based on their meaning; entire words (such as "the") are eliminated, the ordering is not deterministic. This usually requires special software, and is highly dependent on the intended use.

While the historical collation order formally is at level 1, for the English language it corresponds roughly to elements at level 2. The user expects to see the output from the `ls` utility sorted very much as as it would be in a dictionary. While telephone book ordering would be an optimal goal for standard collation, this was ruled out as the order would be language dependent. Furthermore, a requirement was that the order must be determined solely from the text string and the

collation rules; no external information (e.g., “pronunciation dictionaries”) could be required.

As a result, the goal for the collation support is at level 3. This also matches the requirements for the proposed Canadian collation order, as well as other, known collation requirements for alphabetic scripts. It specifically rules out collation based on pronunciation rules, or based on semantic analysis of the text.

The syntax for the LC_COLLATE category source is the result of a cooperative effort between representatives for many countries and organizations working with international issues, such as UniForum, X/Open, and ISO, and it meets the requirements for level 3, and has been verified to produce the correct result with examples based on French, Canadian, and Danish collation order, as well as meeting the requirements in the X/Open Portability Guide, Issue 3. {B31}. Because it supports multicharacter collating elements, it is also capable of supporting collation in code sets where a character is expressed using nonspacing characters followed by the base character (such as ISO 6937 {B6}).

The directives that can be specified in an operand to the `order_start` keyword are based on the requirements specified in several proposed standards and in customary use. The following is a rephrasing of rules defined for “lexical ordering in English and French” by the Canadian Standards Association (text in brackets is rephrased):

- (1) Once special characters ([punctuation]) have been removed from original strings, the ordering is determined by scanning forward (left to right) [disregarding case and diacriticals].
- (2) In case of equivalence, special characters are once again removed from original strings and the ordering is determined scanning backward (starting from the rightmost character of the string and back), character by character, [disregarding case but considering diacriticals].
- (3) In case of repeated equivalence, special characters are removed again from original strings and the ordering is determined scanning forward, character by character, [considering both case and diacriticals].
- (4) If there is still an ordering equivalence after rules (1) through (3) have been applied, then only special characters and the position they occupy in the string are considered to determine ordering. The string that has a special character in the lowest position comes first. If two strings have a special character in the same position, the character [with the lowest collation value] comes first. In case of equality, the other special characters are considered until there is a difference or all special characters have been exhausted.

It is estimated that the standard covers the requirements for all European languages, and no particular problems are anticipated with Slavic or Middle East character sets.

The Far East (particularly Japanese/Chinese) collations are often based on contextual information and pronunciation rules (the same ideogram can have different meanings and different pronunciations). Such collation, in general, falls

outside the desired goal of the standard. There are, however, several other collation rules (stroke/radical, or “most common pronunciation”) which can be supported with the mechanism described here.

Previous drafts contained a substitute statement, which performed a regular expression style replacement before string compares. It has been withdrawn based on balloter objections that it was not required for the types of ordering POSIX.2 is aimed at.

The character (and collating element) order is defined by the order in which characters and elements are specified between the `order_start` and `order_end` keywords. This character order is used in range expressions in regular expressions (see 2.8). Weights assigned to the characters and elements defines the collation sequence; in the absence of weights, the character order is also the collation sequence.

The `position` keyword was introduced to provide the capability to consider, in a compare, the relative position of non-IGNORED characters. As an example, consider the two strings “o-ring” and “or-ing”. Assuming the hyphen is IGNORED on the first pass, the two strings will compare equal, and the position of the hyphen is immaterial. On second pass, all characters except the hyphen are IGNORED, and in the normal case the two strings would again compare equal. By taking position into account, the first collates before the second.

2.5.2.3 LC_MONETARY

The LC_MONETARY category shall define the rules and symbols that shall be used to format monetary numeric information. The operands are strings. For some keywords, the strings can contain only integers. Keywords that are not provided, string values set to the empty string (“”), or integer keywords set to -1, shall be used to indicate that the value is unspecified. The following keywords shall be recognized:

<p><code>copy</code></p>	Specify the name of an existing locale to be used as the source for the definition of this category. If this keyword is specified, no other keyword shall be specified.
--------------------------	---

<p><code>int_curr_symbol</code></p>	The international currency symbol. The operand shall be a four-character string, with the first three characters containing the alphabetic international currency symbol in accordance with those specified in ISO 4217 {3} (<i>Codes for the representation of currencies and funds</i>). The fourth character shall be the character used to separate the international currency symbol from the monetary quantity.
-------------------------------------	---

<p><code>currency_symbol</code></p>	The string that shall be used as the local currency symbol.
-------------------------------------	---

<p><code>mon_decimal_point</code></p>	The operand is a string containing the symbol that shall be used as the decimal delimiter in monetary formatted quantities. In contexts where other standards
---------------------------------------	---

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

Table 2-8 – LC_MONETARY Category Definition in the POSIX Locale

2449	Table 2-8 – LC_MONETARY Category Definition in the POSIX Locale		
2450	<hr/>		
2451	LC_MONETARY		
2452	# This is the POSIX Locale definition for		
2453	# the LC_MONETARY category.		
2454	#		
2455	int_curr_symbol	" "	
2456	currency_symbol	" "	
2457	mon_decimal_point	" "	
2458	mon_thousands_sep	" "	
2459	mon_grouping	" "	
2460	positive_sign	" "	
2461	negative_sign	" "	
2462	int_frac_digits	-1	
2463	p_cs_precedes	-1	
2464	p_sep_by_space	-1	
2465	n_cs_precedes	-1	
2466	n_sep_by_space	-1	
2467	p_sign_posn	-1	
2468	n_sign_posn	-1	
2469	#		
2470	END LC_MONETARY		
2471	<hr/>		
2472		limit the mon_decimal_point to a single byte, the	2
2473		result of specifying a multibyte operand is unspecified.	2
2474	mon_thousands_sep	The operand is a string containing the symbol that	2
2475		shall be used as a separator for groups of digits to the	2
2476		left of the decimal delimiter in formatted monetary	2
2477		quantities. In contexts where other standards limit	2
2478		the mon_thousands_sep to a single byte, the result of	2
2479		specifying a multibyte operand is unspecified.	2
2480	mon_grouping	Define the size of each group of digits in formatted	
2481		monetary quantities. The operand is a sequence of	
2482		integers separated by semicolons. Each integer	
2483		specifies the number of digits in each group, with the	
2484		initial integer defining the size of the group immedi-	
2485		ately preceding the decimal delimiter, and the follow-	
2486		ing integers defining the preceding groups. If the last	2
2487		integer is not -1, then the size of the previous group (if	2
2488		any) shall be repeatedly used for the remainder of the	2
2489		digits. If the last integer is -1, then no further group-	2
2490		ing shall be performed.	2
2491	positive_sign	A string that shall be used to indicate a nonnegative-	
2492		valued formatted monetary quantity.	
2493	negative_sign	A string that shall be used to indicate a negative-	
2494		valued formatted monetary quantity.	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2495	<code>int_frac_digits</code>	An integer representing the number of fractional digits
2496		(those to the right of the decimal delimiter) to be writ-
2497		ten in a formatted monetary quantity using
2498		<code>int_curr_symbol</code> .
2499	<code>frac_digits</code>	An integer representing the number of fractional digits
2500		(those to the right of the decimal delimiter) to be writ-
2501		ten in a formatted monetary quantity using
2502		<code>currency_symbol</code> .
2503	<code>p_cs_precedes</code>	An integer set to 1 if the <code>currency_symbol</code> or
2504		<code>int_curr_symbol</code> precedes the value for a nonnega-
2505		tive formatted monetary quantity, and set to 0 if the
2506		symbol succeeds the value.
2507	<code>p_sep_by_space</code>	An integer set to 0 if no space separates the
2508		<code>currency_symbol</code> or <code>int_curr_symbol</code> from the
2509		value for a nonnegative formatted monetary quantity,
2510		set to 1 if a space separates the symbol from the value,
2511		and set to 2 if a space separates the symbol and the
2512		sign string, if adjacent.
2513	<code>n_cs_precedes</code>	An integer set to 1 if the <code>currency_symbol</code> or
2514		<code>int_curr_symbol</code> precedes the value for a negative
2515		formatted monetary quantity, and set to 0 if the sym-
2516		bol succeeds the value.
2517	<code>n_sep_by_space</code>	An integer set to 0 if no space separates the
2518		<code>currency_symbol</code> or <code>int_curr_symbol</code> from the
2519		value for a negative formatted monetary quantity, set
2520		to 1 if a space separates the symbol from the value,
2521		and set to 2 if a space separates the symbol and the
2522		sign string, if adjacent.
2523	<code>p_sign_posn</code>	An integer set to a value indicating the positioning of
2524		the <code>positive_sign</code> for a nonnegative formatted
2525		monetary quantity. The following integer values shall
2526		be recognized:
2527		0 Parentheses enclose the quantity and the
2528		<code>currency_symbol</code> or <code>int_curr_symbol</code> .
2529		1 The sign string precedes the quantity and the
2530		<code>currency_symbol</code> or <code>int_curr_symbol</code> .
2531		2 The sign string succeeds the quantity and the
2532		<code>currency_symbol</code> or <code>int_curr_symbol</code> .
2533		3 The sign string immediately precedes the
2534		<code>currency_symbol</code> or <code>int_curr_symbol</code> .
2535		4 The sign string immediately succeeds the
2536		<code>currency_symbol</code> or <code>int_curr_symbol</code> .

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2537 `n_sign_posn` An integer set to a value indicating the positioning of
 2538 the `negative_sign` for a negative formatted mone- 1
 2539 tary quantity. The following integer values shall be
 2540 recognized:

- 2541 0 Parentheses enclose the quantity and the
- 2542 `currency_symbol` or `int_curr_symbol`.
- 2543 1 The sign string precedes the quantity and the
- 2544 `currency_symbol` or `int_curr_symbol`.
- 2545 2 The sign string succeeds the quantity and the
- 2546 `currency_symbol` or `int_curr_symbol`.
- 2547 3 The sign string immediately precedes the
- 2548 `currency_symbol` or `int_curr_symbol`.
- 2549 4 The sign string immediately succeeds the
- 2550 `currency_symbol` or `int_curr_symbol`.

2551 **2.5.2.3.1 LC_MONETARY Rationale.** *(This subclause is not a part of P1003.2)*

2552 The currency symbol does not appear in LC_MONETARY because it is not defined
 2553 in the C Standard's {7} C locale.

2554 The C Standard {7} limits the size of decimal points and thousands delimiters to 2
 2555 single-byte values. In locales based on multibyte coded character sets this cannot 2
 2556 be enforced, obviously; this standard does not prohibit such characters, but makes 2
 2557 the behavior unspecified [in the text "In contexts where other standards ..."]. 2

2558 The grouping specification is based on, but not identical to, the C Standard {7}. 2
 2559 The "-1" signals that no further grouping shall be performed, the equivalent of 2
 2560 {CHAR_MAX} in the C Standard {7}). 2

2561 The locale definition is an extension of the C Standard {7} *localeconv()*
 2562 specification. In particular, rules on how `currency_symbol` is treated are
 2563 extended to also cover `int_curr_symbol`, and `p_set_by_space` and
 2564 `n_sep_by_space` have been augmented with the value 2, which places a space
 2565 between the sign and the symbol (if they are adjacent; otherwise it should be
 2566 treated as a 0). The following table shows the result of various combinations:

		p_sep_by_space		
		2	1	0
2569	p_cs_precedes = 1	p_sign_posn = 0 (\$1.25)	(\$ 1.25)	(\$1.25)
2570		p_sign_posn = 1 + \$1.25	+\$ 1.25	+\$1.25
2571		p_sign_posn = 2 \$1.25 +	\$ 1.25+	\$1.25+
2572		p_sign_posn = 3 + \$1.25	+\$ 1.25	+\$1.25
2573		p_sign_posn = 4 \$ +1.25	\$+ 1.25	\$+1.25
2574	p_cs_precedes = 0	p_sign_posn = 0 (1.25 \$)	(1.25 \$)	(1.25\$)
2575		p_sign_posn = 1 +1.25 \$	+1.25 \$	+1.25\$
2576		p_sign_posn = 2 1.25\$ +	1.25 \$+	1.25\$+
2577		p_sign_posn = 3 1.25+ \$	1.25 +\$	1.25+\$
2578		p_sign_posn = 4 1.25\$ +	1.25 \$+	1.25\$+

2579 The following is an example of the interpretation of the `mon_grouping` keyword.
 2580 Assuming that the value to be formatted is 123456789 and the
 2581 `mon_thousands_sep` is ', then the following table shows the result. The third
 2582 column shows the equivalent C Standard {7} string that would be used to accom-
 2583 modate this grouping. It is the responsibility of the utility to perform mappings of
 2584 the formats in this clause to those used by language bindings such as the
 2585 C Standard {7}.

2586	<u>mon_grouping</u>	<u>Formatted Value</u>	<u>C Standard {7} String</u>	1
2587	3;-1	123456'789	"\3\177"	2
2588	3	123'456'789	"\3"	2
2589	3;2;-1	1234'56'789	"\3\2\177"	2
2590	3;2	12'34'56'789	"\3\2"	2
2591	-1	123456789	"177"	2

2592 In these examples, the octal value of {CHAR_MAX} is 177.

2593 2.5.2.4 LC_NUMERIC

2594 The LC_NUMERIC category shall define the rules and symbols that shall be used
 2595 to format nonmonetary numeric information. The operands are strings. For some
 2596 keywords, the strings only can contain integers. Keywords that are not provided,
 2597 string values set to the empty string (""), or integer keywords set to -1, shall be
 2598 used to indicate that the value is unspecified. The following keywords shall be
 2599 recognized:

2600	<code>copy</code>	Specify the name of an existing locale to be used as the	
2601		source for the definition of this category. If this keyword	
2602		is specified, no other keyword shall be specified.	
2603	<code>decimal_point</code>	The operand is a string containing the symbol that shall	2
2604		be used as the decimal delimiter in numeric, nonmonetary	2
2605		formatted quantities. This keyword cannot be omitted	2
2606		and cannot be set to the empty string. In contexts where	2
2607		other standards limit the <code>decimal_point</code> to a single	2
2608		byte, the result of specifying a multibyte operand is	2

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2609		unspecified.	2
2610	thousands_sep	The operand is a string containing the symbol that shall	2
2611		be used as a separator for groups of digits to the left of the	2
2612		decimal delimiter in numeric, nonmonetary formatted	2
2613		monetary quantities. In contexts where other standards	2
2614		limit the thousands_sep to a single byte, the result of	2
2615		specifying a multibyte operand is unspecified.	2
2616	grouping	Define the size of each group of digits in formatted non-	
2617		monetary quantities. The operand is a sequence of	
2618		integers separated by semicolons. Each integer specifies	
2619		the number of digits in each group, with the initial integer	
2620		defining the size of the group immediately preceding the	
2621		decimal delimiter, and the following integers defining the	
2622		preceding groups. If the last integer is not -1, then the	2
2623		size of the previous group (if any) shall be repeatedly used	2
2624		for the remainder of the digits. If the last integer is -1,	2
2625		then no further grouping shall be performed.	2

Table 2-9 – LC_NUMERIC Category Definition in the POSIX Locale

2628	LC_NUMERIC		
2629	# This is the POSIX Locale definition for		
2630	# the LC_NUMERIC category.		
2631	#		
2632	decimal_point	"<period>"	2
2633	thousands_sep	" "	
2634	grouping	0	
2635	#		
2636	END LC_NUMERIC		

2638 **2.5.2.4.1 LC_NUMERIC Rationale.** *(This subclause is not a part of P1003.2)*

2639 See the rationale for LC_MONETARY (2.5.2.3.1) for a description of the behavior of
 2640 grouping. 1

2641 **2.5.2.5 LC_TIME**

2642 The LC_TIME category shall define the interpretation of the field descriptors sup-
 2643 ported by the date utility (see 4.15).

2644 The following mandatory keywords shall be recognized:

2645	copy	Specify the name of an existing locale to be used as the source for	
2646		the definition of this category. If this keyword is specified, no	
2647		other keyword shall be specified.	
2648	abday	Define the abbreviated weekday names, corresponding to the %a	
2649		field descriptor. The operand shall consist of seven semicolon-	
2650		separated strings. The first string shall be the abbreviated name	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

Table 2-10 – LC_TIME Category Definition in the POSIX Locale

```

2651
2652
2653 LC_TIME
2654 # This is the POSIX Locale definition for
2655 # the LC_TIME category.
2656 #
2657 # Abbreviated weekday names (%a)
2658 abday    "<S><u><n>" ; "<M><o><n>" ; "<T><u><e>" ; "<W><e><d>" ; \
2659          "<T><h><u>" ; "<F><r><i>" ; "<S><a><t>"
2660 #
2661 # Full weekday names (%A)
2662 day      "<S><u><n><d><a><y>" ; "<M><o><n><d><a><y>" ; \
2663          "<T><u><e><s><d><a><y>" ; "<W><e><d><n><e><s><d><a><y>" ; \
2664          "<T><h><u><r><s><d><a><y>" ; "<F><r><i><d><a><y>" ; \
2665          "<S><a><t><u><r><d><a><y>"
2666 #
2667 # Abbreviated month names (%b)
2668 abmon    "<J><a><n>" ; "<F><e><b>" ; "<M><a><r>" ; \
2669          "<A><p><r>" ; "<M><a><y>" ; "<J><u><n>" ; \
2670          "<J><u><l>" ; "<A><u><g>" ; "<S><e><p>" ; \
2671          "<O><c><t>" ; "<N><o><v>" ; "<D><e><c>"
2672 #
2673 # Full month names (%B)
2674 mon      "<J><a><n><u><a><r><y>" ; "<F><e><b><r><u><a><r><y>" ; \
2675          "<M><a><r><c><h>" ; "<A><p><r><i><l>" ; \
2676          "<M><a><y>" ; "<J><u><n><e>" ; \
2677          "<J><u><l><y>" ; "<A><u><g><u><s><t>" ; \
2678          "<S><e><p><t><e><m><b><e><r>" ; "<O><c><t><o><b><e><r>" ; \
2679          "<N><o><v><e><m><b><e><r>" ; "<D><e><c><e><m><b><e><r>"
2680 #
2681 # Equivalent of AM/PM (%p)      "AM" ; "PM"
2682 am_pm    "<A><M>" ; "<P><M>"
2683 #
2684 # Appropriate date and time representation (%c)
2685 #      "%a %b %e %H:%M:%S %Y"
2686 d_t_fmt  "<percent-sign><a><space><percent-sign><b><space><percent-sign><e>\
2687          <space><percent-sign><H><colon><percent-sign><M>\
2688          <colon><percent-sign><S><space><percent-sign><Y>"
2689 #
2690 # Appropriate date representation (%x)      "%m/%d/%y"
2691 d_fmt    "<percent-sign><m><slash><percent-sign><d><slash><percent-sign><y>"
2692 #
2693 # Appropriate time representation (%X)      "%H:%M:%S"
2694 t_fmt    "<percent-sign><H><colon><percent-sign><M><colon><percent-sign><S>"
2695 #
2696 # Appropriate 12-hour time representation (%r) "%I:%M:%S %p"
2697 t_fmt_ampm "<percent-sign><I><colon><percent-sign><M><colon>\
2698          <percent-sign><S> <percent_sign><p>"
2699 #
2700 END LC_TIME
2701

```

2702		of the first day of the week (Sunday), the second the abbreviated	
2703		name of the second day, and so on.	
2704	day	Define the full weekday names, corresponding to the %A field	
2705		descriptor. The operand shall consist of seven semicolon-separated	
2706		strings. The first string shall be the full name of the first day of	
2707		the week (Sunday), the second the full name of the second day, and	
2708		so on.	
2709	abmon	Define the abbreviated month names, corresponding to the %b field	
2710		descriptor. The operand shall consist of twelve semicolon-separated	
2711		strings. The first string shall be the abbreviated name	
2712		of the first month of the year (January), the second the abbreviated	
2713		name of the second month, and so on.	
2714	mon	Define the full month names, corresponding to the %B field descrip-	
2715		tor. The operand shall consist of twelve semicolon-separated	
2716		strings. The first string shall be the full name of the first month of	
2717		the year (January), the second the full name of the second month,	
2718		and so on.	
2719	d_t_fmt	Define the appropriate date and time representation, correspond-	
2720		ing to the %c field descriptor. The operand shall consist of a string,	
2721		and can contain any combination of characters and field descrip-	
2722		tors. In addition, the string can contain escape sequences defined	
2723		in Table 2-15.	1
2724	d_fmt	Define the appropriate date representation, corresponding to the	
2725		%x field descriptor. The operand shall consist of a string, and can	
2726		contain any combination of characters and field descriptors. In	
2727		addition, the string can contain escape sequences defined in	1
2728		Table 2-15.	1
2729	t_fmt	Define the appropriate time representation, corresponding to the	
2730		%X field descriptor. The operand shall consist of a string, and can	
2731		contain any combination of characters and field descriptors. In	
2732		addition, the string can contain escape sequences defined in	1
2733		Table 2-15.	1
2734	am_pm	Define the appropriate representation of the <i>ante meridiem</i> and	
2735		<i>post meridiem</i> strings, corresponding to the %p field descriptor.	
2736		The operand shall consist of two strings, separated by a semicolon.	
2737		The first string shall represent the <i>ante meridiem</i> designation, the	
2738		last string the <i>post meridiem</i> designation.	
2739	t_fmt_ampm		
2740		Define the appropriate time representation in the 12-hour clock	
2741		format with am_pm, corresponding to the %r field descriptor. The	
2742		operand shall consist of a string and can contain any combination	
2743		of characters and field descriptors. If the string is empty, the 12-	
2744		hour format is not supported in the locale.	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2745 It is implementation defined whether the following optional keywords shall be
 2746 recognized. If they are not supported, but present in a `localedef` source, they
 2747 shall be ignored.

2748 `era` Shall be used to define alternate Eras, corresponding to the `%E`
 2749 field descriptor modifier. The format of the operand is
 2750 unspecified, but shall support the definition of the `%EC` and
 2751 `%EY` field descriptors, and may also define the `era_year` for-
 2752 mat (`%EY`).

2753 `era_year` Shall be used to define the format of the year in alternate Era
 2754 format, corresponding to the `%EY` field descriptor.

2755 `era_d_fmt` Shall be used to define the format of the date in alternate Era
 2756 notation, corresponding to the `%Ex` field descriptor.

2757 `alt_digits` Shall be used to define alternate symbols for digits,
 2758 corresponding to the `%O` field descriptor modifier. The operand
 2759 shall consist of semicolon-separated strings. The first string
 2760 shall be the alternate symbol corresponding with zero, the
 2761 second string the symbol corresponding with one, and so on.
 2762 Up to 100 alternate symbol strings can be specified. The `%O`
 2763 modifier indicates that the string corresponding to the value
 2764 specified via the field descriptor shall be used instead of the
 2765 value.

2766 **2.5.2.5.1 LC_TIME Rationale.** *(This subclause is not a part of P1003.2)*

2767 Although certain of the field descriptors in the POSIX Locale (such as the name of
 2768 the month) are shown with initial capital letters, this need not be the case in
 2769 other locales. Programs using these fields may need to adjust the capitalization if
 2770 the output is going to be used at the beginning of a sentence.

2771 The `LC_TIME` descriptions of `abday`, `daya`, and `abmon` imply a Gregorian style 1
 2772 calendar (7-day weeks, 12-month years, leap years, etc.). Formatting time strings 1
 2773 for other types of calendars is outside the scope of this standard. 1

2774 As specified under the `date` command, the field descriptors corresponding to the
 2775 optional keywords consist of a modifier followed by a traditional field descriptor
 2776 (for instance `%Ex`). If the optional keywords are not supported by the implementa-
 2777 tion or are unspecified for the current locale, these field descriptors shall be
 2778 treated as the traditional field descriptor. For instance, assume the following key-
 2779 words:

2780	<code>alt_digits</code>	<code>"0th"; "1st"; "2nd"; "3rd"; "4th"; "5th"; \</code>	1
2781		<code>"6th"; "7th"; "8th"; "9th"; "10th"</code>	1
2782	<code>d_fmt</code>	<code>"The %Od day of %B in %Y"</code>	1

2783 On 7/4/1776, the `%x` field descriptor would result in "The 4th day of July in 1776," 1
 2784 while 7/14/1789 would come out as "The 14 day of July in 1789." It can be noted
 2785 that the above example is for illustrative purposes only; the `%O` modifier is pri-
 2786 marily intended to provide for Kanji or Hindi digits in date formats.

2787 While it is clear that an alternate year format is required, there is no consensus
 2788 on the format or the requirements. As a result, while these keywords are
 2789 reserved, the details are left unspecified. It is expected that National Standards
 2790 Bodies will provide specifications.

2791 **2.5.2.6 LC_MESSAGES**

2792 The LC_MESSAGES category shall define the format and values for affirmative
 2793 and negative responses. The operands shall be strings or extended regular
 2794 expressions; see 2.8.4. The following keywords shall be recognized:

2795 copy Specify the name of an existing locale to be used as the source for
 2796 the definition of this category. If this keyword is specified, no
 2797 other keyword shall be specified.

2798 yesexpr The operand shall consist of an extended regular expression that
 2799 describes the acceptable affirmative response to a question expect-
 2800 ing an affirmative or negative response.

2801 noexpr The operand shall consist of an extended regular expression that
 2802 describes the acceptable negative response to a question expecting
 2803 an affirmative or negative response.

2804 **Table 2-11 – LC_MESSAGES Category Definition in the POSIX Locale**

```

2806 LC_MESSAGES
2807 # This is the POSIX Locale definition for
2808 # the LC_MESSAGES category.
2809 #
2810 yesexpr "<circumflex><left-square-bracket><y><Y><right-square-bracket>"
2811 #
2812 noexpr  "<circumflex><left-square-bracket><n><N><right-square-bracket>"
2813 END LC_MESSAGES
  
```

2815 **2.5.2.6.1 LC_MESSAGES Rationale.** *(This subclause is not a part of P1003.2)*

2816 The LC_MESSAGES category is described in 2.6 as affecting the language used by
 2817 utilities for their output. The mechanism used by the implementation to accom-
 2818 plish this, other than the responses shown here in the locale definition file, is not
 2819 specified by this version of this standard. The POSIX.1 working group is develop-
 2820 ing an interface that would allow applications (and, presumably some of the stan-
 2821 dard utilities) to access messages from various message catalogs, tailored to a
 2822 user's LC_MESSAGES value.

2823	2.5.3 Locale Definition Grammar	1
2824	The grammar and lexical conventions in this subclause shall together describe the	1
2825	syntax for the locale definition source. The general conventions for this style of	1
2826	grammar are described in 2.1.2. Any discrepancies found between this grammar	1
2827	and other descriptions in this clause shall be resolved in favor of this grammar.	1
2828	2.5.3.1 Locale Lexical Conventions	1
2829	The lexical conventions for the locale definition grammar are described in this	1
2830	subclause.	1
2831	The following tokens shall be processed (in addition to those string constants	1
2832	shown in the grammar):	1
2833	LOC_NAME A string of characters representing the name of a locale.	1
2834	CHAR Any single character.	1
2835	NUMBER A decimal number, represented by one or more decimal digits.	2
2836	COLLSYMBOL A symbolic name, enclosed between angle brackets. The	1
2837	string shall not duplicate any charmap symbol defined in the	1
2838	current charmap (if any), or a COLLELEMENT symbol.	1
2839	COLLELEMENT A symbolic name, enclosed between angle brackets, which	1
2840	shall not duplicate either any charmap symbol or a CHARSYM-	1
2841	BOL symbol.	1
2842	CHARSYMBOL A symbolic name, enclosed between angle brackets, from the	1
2843	current charmap (if any).	1
2844	OCTAL_CHAR One or more octal representations of the encoding of each byte	1
2845	in a single character. The octal representation consists of an	1
2846	escape_char (normally a backslash) followed by two or more	1
2847	octal digits.	1
2848	HEX_CHAR One or more hexadecimal representations of the encoding of	1
2849	each byte in a single character. The hexadecimal representa-	1
2850	tion consists of an escape_char followed by the constant 'x'	1
2851	and two or more hexadecimal digits.	1
2852	DECIMAL_CHAR One or more decimal representations of the encoding of each	1
2853	byte in a single character. The decimal representation cons-	1
2854	sists of an escape_char and followed by a 'd' and two or	1
2855	more decimal digits.	1
2856	ELLIPSIS The string "...".	1
2857	EXTENDED_REG_EXP	1
2858	An extended regular expression as defined in the grammar in	1
2859	2.8.5.2.	1


```

2902 ctype_hdr           : 'LC_CTYPE' EOL                2
2903                     ;                                2
2904 ctype_keywords       : ctype_keywords ctype_keyword    2
2905                     | ctype_keyword                    2
2906                     ;                                1
2907 ctype_keyword        : charclass_keyword charclass_list EOL 1
2908                     | charconv_keyword charconv_list EOL 1
2909                     ;                                1
2910 charclass_keyword     : 'upper' | 'lower' | 'alpha' | 'digit' 1
2911                     | 'alnum' | 'xdigit' | 'space' | 'print' 1
2912                     | 'graph' | 'blank' | 'cntrl'         1
2913                     ;                                1
2914 charclass_list        : charclass_list ';' char_symbol    2
2915                     | charclass_list ';' ELLIPSIS ';' char_symbol 1
2916                     | char_symbol                        2
2917                     ;                                1
2918 charconv_keyword      : 'toupper'                       1
2919                     | 'tolower'                         1
2920                     ;                                1
2921 charconv_list         : charconv_list ';' charconv_entry 2
2922                     | charconv_entry                    2
2923                     ;                                1
2924 charconv_entry        : '(' char_symbol ',' char_symbol ')' 1
2925                     ;                                1
2926 ctype_tlr            : 'END' 'LC_CTYPE' EOL            2
2927                     ;                                1
2928 /*      The following is the LC_COLLATE category grammar */ 1
2929 lc_collate            : collate_hdr collate_keywords      collate_tlr 2
2930                     | collate_hdr 'copy' locale_name EOL collate_tlr 2
2931                     ;                                2
2932 collate_hdr           : 'LC_COLLATE' EOL                2
2933                     ;                                2
2934 collate_keywords      :                                2
2935                     | opt_statements order_statements    2
2936                     ;                                1
2937 opt_statements        : opt_statements collating_symbols 2
2938                     | opt_statements collating_elements 2
2939                     | collating_symbols                  1
2940                     | collating_elements                 1
2941                     ;                                1
2942 collating_symbols     : 'collating-symbol' COLLSYMBOL EOL 1
2943                     ;                                1
2944 collating_elements    : 'collating-element' COLLELEMENT 1
2945                     'from' '"' char_list '"' EOL        2
2946                     ;                                1

```


2947	order_statements	: order_start collation_order order_end	1
2948		;	1
2949	order_start	: 'order_start' EOL	1
2950		'order_start' order_opts EOL	1
2951		;	1
2952	order_opts	: order_opts ';' order_opt	2
2953		order_opt	2
2954		;	1
2955	order_opt	: order_opt ',' opt_word	2
2956		opt_word	2
2957		;	1
2958	opt_word	: 'forward' 'backward' 'position'	2
2959		;	1
2960	collation_order	: collation_order collation_entry	2
2961		collation_entry	2
2962		;	1
2963	collation_entry	: COLLSYMBOL EOL	1
2964		collation_element weight_list EOL	1
2965		collation_element EOL	2
2966		;	1
2967	collation_element	: char_symbol	1
2968		COLLELEMENT	1
2969		ELLIPSIS	1
2970		'UNDEFINED'	1
2971		;	1
2972	weight_list	: weight_list ';' weight_symbol	2
2973		weight_list ';'	2
2974		weight_symbol	2
2975		;	1
2976	weight_symbol	: char_symbol	2
2977		COLLSYMBOL	1
2978		'"' char_list '"'	1
2979		ELLIPSIS	1
2980		'IGNORE'	1
2981		;	1
2982	order_end	: 'order_end' EOL	1
2983		;	1
2984	collate_tlr	: 'END' 'LC_COLLATE' EOL	2
2985		;	1
2986	/* The following is the LC_MESSAGES category grammar */		1
2987	lc_messages	: messages_hdr messages_keywords messages_tlr	2
2988		messages_hdr 'copy' locale_name EOL messages_tlr	2
2989		;	2
2990	messages_hdr	: 'LC_MESSAGES' EOL	2
2991		;	2

```

2992 messages_keywords      : messages_keywords messages_keyword      2
2993                          | messages_keyword                        2
2994                          ;                                          1
2995 messages_keyword         : 'yesexpr' ' "' EXTENDED_REG_EXP ' "' EOL  2
2996                          | 'noexpr' ' "' EXTENDED_REG_EXP ' "' EOL  2
2997                          ;                                          2
2998 messages_tlr             : 'END' 'LC_MESSAGES' EOL                  2
2999                          ;                                          1
3000 /*      The following is the LC_MONETARY category grammar */        1
3001 lc_monetary              : monetary_hdr monetary_keywords      monetary_tlr  2
3002                          | monetary_hdr 'copy' locale_name EOL  monetary_tlr  2
3003                          ;                                          2
3004 monetary_hdr             : 'LC_MONETARY' EOL                      2
3005                          ;                                          2
3006 monetary_keywords        : monetary_keywords monetary_keyword  2
3007                          | monetary_keyword                      2
3008                          ;                                          1
3009 monetary_keyword         : mon_keyword_string mon_string EOL      1
3010                          | mon_keyword_char NUMBER EOL            2
3011                          | mon_keyword_char '-1' EOL              2
3012                          | mon_keyword_grouping mon_group_list EOL  1
3013                          ;                                          1
3014 mon_keyword_string       : 'int_curr_symbol' | 'currency_symbol'    1
3015                          | 'mon_decimal_point' | 'mon_thousands_sep'  1
3016                          | 'positive_sign' | 'negative_sign'         1
3017                          ;                                          1
3018 mon_string               : ' "' char_list ' "'                    1
3019                          | ' "' '                                     1
3020                          ;                                          1
3021 mon_keyword_char         : 'int_frac_digits' | 'frac_digits'       1
3022                          | 'p_cs_precedes' | 'p_sep_by_space'       1
3023                          | 'n_cs_precedes' | 'n_sep_by_space'       1
3024                          | 'p_sign_posn' | 'n_sign_posn'            1
3025                          ;                                          1
3026 mon_keyword_grouping     : 'mon_grouping'                          1
3027                          ;                                          1
3028 mon_group_list           : NUMBER                                  2
3029                          | mon_group_list ';' NUMBER                2
3030                          ;                                          2
3031 monetary_tlr             : 'END' 'LC_MONETARY' EOL                  2
3032                          ;                                          2
3033 /*      The following is the LC_NUMERIC category grammar */        2
3034 lc_numeric               : numeric_hdr numeric_keywords      numeric_tlr  2
3035                          | numeric_hdr 'copy' locale_name EOL  numeric_tlr  2
3036                          ;                                          2

```

3037	numeric_hdr	: 'LC_NUMERIC' EOL	2
3038		;	2
3039	numeric_keywords	: numeric_keywords numeric_keyword	2
3040		numeric_keyword	2
3041		;	1
3042	numeric_keyword	: num_keyword_string num_string EOL	1
3043		num_keyword_grouping num_group_list EOL	1
3044		;	1
3045	num_keyword_string	: 'decimal_point'	1
3046		'thousands_sep'	1
3047		;	1
3048	num_string	: '"' char_list '"'	1
3049		'"""'	1
3050		;	1
3051	num_keyword_grouping	: 'num_grouping'	1
3052		;	1
3053	num_group_list	: NUMBER	2
3054		num_group_list ';' NUMBER	2
3055		;	1
3056	numeric_tlr	: 'END' 'LC_NUMERIC' EOL	2
3057		;	1
3058	/* The following is the LC_TIME category grammar */		1
3059	lc_time	: time_hdr time_keywords time_tlr	2
3060		time_hdr 'copy' locale_name EOL time_tlr	2
3061		;	1
3062	time_hdr	: 'LC_TIME' EOL	2
3063		;	1
3064	time_keywords	: time_keywords time_keyword	2
3065		time_keyword	2
3066		;	1
3067	time_keyword	: time_keyword_name time_list EOL	2
3068		time_keyword_fmt time_string EOL	1
3069		time_keyword_opt time_list EOL	1
3070		;	1
3071	time_keyword_name	: 'abday' 'day' 'abmon' 'mon'	2
3072		;	1
3073	time_keyword_fmt	: 'd_t_fmt' 'd_fmt' 't_fmt' 'am_pm' 't_fmt_ampm'	1
3074		;	1
3075	time_keyword_opt	: 'era' 'era_year' 'era_d_fmt' 'alt_digits'	1
3076		;	1
3077	time_list	: time_list ';' time_string	2
3078		time_string	2
3079		;	1
3080	time_string	: '"' char_list '"'	1
3081		;	1

```

3082 time_tlr          : 'END' 'LC_TIME' EOL      2
3083                  ;                          1

```

3084 **2.5.4 Locale Definition Example.** *(This subclause is not a part of P1003.2)*

3085 The following is an example of a locale definition file that could be used as input
 3086 to the localedef utility. It assumes that the utility is executed with the -f
 3087 option, naming a *charmap* file with (at least) the following content:

```

3088 CHARMAP
3089 <space>      \x20
3090 <dollar>     \x24
3091 <A>          \101
3092 <a>          \141
3093 <A-acute>    \346
3094 <a-acute>    \365
3095 <A-grave>    \300
3096 <a-grave>    \366
3097 <b>          \142
3098 <C>          \103
3099 <c>          \143
3100 <c-cedilla>  \347
3101 <d>          \x64
3102 <H>          \110
3103 <h>          \150
3104 <eszet>     \xb7
3105 <s>          \x73
3106 <z>          \x7a
3107 END CHARMAP

```

3108 It should not be taken as complete or to represent any actual locale, but only to
 3109 illustrate the syntax.

3110 A further set of examples is offered as part of Annex F.

```

3111 #
3112 LC_CTYPE
3113 lower  <a>;<b>;<c>;<c-cedilla>;<d>;...;<z>
3114 upper  A;B;C;Ç;...;Z
3115 space  \x20;\x09;\x0a;\x0b;\x0c;\x0d
3116 blank  \040;\011
3117 toupper (<a>,<A>);(b,B);(c,C);(ç,Ç);(d,D);(z,Z)
3118 END LC_CTYPE
3119 #
3120 LC_COLLATE
3121 #
3122 # The following example of collation is based on the proposed
3123 # Canadian standard Z243.4.1-1990, "Canadian Alphanumeric
3124 # Ordering Standard For Character sets of CSA Z234.4 Standard".
3125 # (Other parts of this example locale definition file do not
3126 # purport to relate to Canada, or to any other real culture.)
3127 # The proposed standard defines a 4-weight collation, such that
3128 # in the first pass, characters are compared without regard to

```

```

3129 # case or accents; in second pass, backwards compare without
3130 # regard to case; in the third pass, forward compare without
3131 # regard to diacriticals. In the 3 first passes, non-alphabetic      2
3132 # characters are ignored; in the fourth pass, only special
3133 # characters are considered, such that "The string that has a
3134 # special character in the lowest position comes first. If two
3135 # strings have a special character in the same position, the
3136 # collation value of the special character determines ordering.
3137 #
3138 # Only a subset of the character set is used here; mostly to
3139 # illustrate the set-up.
3140 #
3141 #
3142 collating-symbol <LOW_VALUE>
3143 collating-symbol <LOWER-CASE>
3144 collating-symbol <SUBSCRIPT-LOWER>
3145 collating-symbol <SUPERSCRIPT-LOWER>
3146 collating-symbol <UPPER-CASE>
3147 collating-symbol <NO-ACCENT>
3148 collating-symbol <PECULIAR>
3149 collating-symbol <LIGATURE>
3150 collating-symbol <ACUTE>
3151 collating-symbol <GRAVE>
3152 # Further collating-symbols follow.
3153 #
3154 # Properly, the standard does not include any multi-character
3155 # collating elements; the one below is added for completeness.
3156 #
3157 collating_element <ch> from <c><h>
3158 collating_element <CH> from <C><H>
3159 collating_element <Ch> from <C><h>
3160 #
3161 order_start forward;backward;forward;forward,position
3162 #
3163 # Collating symbols are specified first in the sequence to allocate
3164 # basic collation values to them, lower than that of any character.
3165
3166 <LOW_VALUE>
3167 <LOWER-CASE>
3168 <SUBSCRIPT-LOWER>
3169 <SUPERSCRIPT-LOWER>
3170 <UPPER-CASE>
3171 <NO-ACCENT>
3172 <PECULIAR>
3173 <LIGATURE>
3174 <ACUTE>
3175 <GRAVE>
3176 <RING-ABOVE>
3177 <DIAERESIS>
3178 <TILDE>
3179 # Further collating symbols are given a basic collating value here.

```

```

3180 #
3181 # Here follows special characters.
3182 <space>      IGNORE;IGNORE;IGNORE;<space>
3183 # Other special characters follow here.
3184 #
3185 # Here comes the regular characters.
3186 <a>           <a>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
3187 <A>           <a>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
3188 <a-acute>     <a>;<ACUTE>;<LOWER-CASE>;IGNORE
3189 <A-acute>     <a>;<ACUTE>;<UPPER-CASE>;IGNORE
3190 <a-grave>     <a>;<GRAVE>;<LOWER-CASE>;IGNORE
3191 <A-grave>     <a>;<GRAVE>;<UPPER-CASE>;IGNORE
3192 <ae>         <a><e>;<LIGATURE><LIGATURE>;<LOWER-CASE><LOWER-CASE>;IGNORE
3193 <AE>         <a><e>;<LIGATURE><LIGATURE>;<UPPER-CASE><UPPER-CASE>;IGNORE
3194 <b>           <b>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
3195 <B>           <b>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
3196 <c>           <c>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
3197 <C>           <c>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
3198 <ch>         <ch>;<NO-ACCENT>;<LOWER-CASE>;IGNORE
3199 <Ch>         <ch>;<NO-ACCENT>;<PECULIAR>;IGNORE
3200 <CH>         <ch>;<NO-ACCENT>;<UPPER-CASE>;IGNORE
3201 #
3202 # As an example, the strings "Bach" and "bach" could be encoded (for
3203 # compare purposes) as:
3204 # "Bach"      <b>;<a>;<ch>;<LOW_VALUE>;<NO-ACCENT>;<NO-ACCENT>;\      2
3205 #              <NO-ACCENT>;<LOW_VALUE>;<UPPER>;<LOWER>;<LOWER>;<NULL>      2
3206 # "bach"      <b>;<a>;<ch>;<LOW_VALUE>;<NO-ACCENT>;<NO-ACCENT>;\      2
3207 #              <NO-ACCENT>;<LOW_VALUE>;<LOWER>;<LOWER>;<LOWER>;<NULL>      2
3208 #
3209 # The two strings are equal in pass 1 and 2, but differ in pass 3.
3210 #
3211 # Further characters follow.
3212 #
3213 UNDEFINED    IGNORE;IGNORE;IGNORE;IGNORE
3214 #
3215 order_end
3216 #
3217 END LC_COLLATE
3218 #
3219 LC_MONETARY
3220 int_curr_symbol      "USD "
3221 currency_symbol      "$"
3222 mon_decimal_point    "."
3223 mon_grouping         3;0
3224 positive_sign        " "
3225 negative_sign        "- "
3226 p_cs_precedes        1
3227 n_sign_posn          0
3228 END LC_MONETARY
3229 #
3230 LC_NUMERIC
3231 copy "US_en.ASCII"

```

1

```

3232 END LC_NUMERIC
3233 #
3234 LC_TIME
3235 abday      "Sun";"Mon";"Tue";"Wed";"Thu";"Fri";"Sat"
3236 #
3237 day        "Sunday";"Monday";"Tuesday";"Wednesday";\
3238            "Thursday";"Friday";"Saturday"
3239 #
3240 abmon      "Jan";"Feb";"Mar";"Apr";"May";"Jun";\
3241            "Jul";"Aug";"Sep";"Oct";"Nov";"Dec"
3242 #
3243 mon        "January";"February";"March";"April";\
3244            "May";"June";"July";"August";"September";\
3245            "October";"November";"December"
3246 #
3247 d_t_fmt    "%a %b %d %T %Z %Y\n"
3248 END LC_TIME
3249 #
3250 LC_MESSAGES
3251 yesexpr    "^[yY][[:alpha:]]*|(OK)"
3252 #
3253 noexpr     "^[nN][[:alpha:]]*"
3254 END LC_MESSAGES

```

1

1

2.6 Environment Variables

Environment variables defined in this clause affect the operation of multiple utilities 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 subclause of the utility descriptions for information on environment variable usage.

The value of an environment variable is a string of characters, as described in 2.7 in POSIX.1 {8}.

Environment variable names used by the standard utilities shall consist solely of uppercase letters, digits, and the `_` (underscore) from the characters defined in 2.4. The namespace of environment variable names containing lowercase letters shall be reserved for applications. Applications can define any environment variables with names from this namespace without modifying the behavior of the standard utilities.

If the following variables are present in the environment during the execution of an application or utility, they are given the meaning described below. They may be put into the environment, or changed, by either the implementation or the user. If they are defined in the utility's environment, the standard utilities assume they have the specified meaning. Conforming applications shall not set these environment variables to have meanings other than as described. See 7.2 and 3.12 for methods of accessing these variables.

HOME	A pathname of the user's home directory.	
LANG	This variable shall determine the locale category for any category not specifically selected via a variable starting with LC_ . LANG and the LC_ variables can be used by applications to determine the language for messages and instructions, collating sequences, date formats, etc. Additional semantics of this variable, if any, are implementation defined.	1 1 1
LC_ALL	This variable shall override the value of the LANG variable and the value of any of the other variables starting with LC_ .	
LC_COLLATE	This variable shall determine the locale category for character collation information within bracketed regular expressions and for sorting. This environment variable determines the behavior of ranges, equivalence classes, and multicharacter collating elements. Additional semantics of this variable, if any, are implementation defined.	
LC_CTYPE	This variable shall determine the locale category for character handling functions. This environment variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters), the classification of characters (e.g., alpha, digit,	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

3298		graph), and the behavior of character classes. Additional
3299		semantics of this variable, if any, are implementation
3300		defined.
3301	LC_MESSAGES	This variable shall determine the locale category for process-
3302		ing affirmative and negative responses and the language
3303		and cultural conventions in which messages should be writ-
3304		ten. Additional semantics of this variable, if any, are imple-
3305		mentation defined. The language and cultural conventions
3306		of diagnostic and informative messages whose format is
3307		unspecified by this standard should be affected by the set-
3308		ting of LC_MESSAGES .
3309	LC_MONETARY	This variable shall determine the locale category for
3310		monetary-related numeric formatting information. Addi-
3311		tional semantics of this variable, if any, are implementation
3312		defined.
3313	LC_NUMERIC	This variable shall determine the locale category for
3314		numeric formatting (for example, thousands separator and
3315		radix character) information. Additional semantics of this
3316		variable, if any, are implementation defined.
3317	LC_TIME	This variable shall determine the locale category for date
3318		and time formatting information. Additional semantics of
3319		this variable, if any, are implementation defined.
3320	LOGNAME	The user's login name.
3321	PATH	The sequence of path prefixes that certain functions and
3322		utilities apply in searching for an executable file known only
3323		by a filename. The prefixes shall be separated by a colon (:).
3324		When a nonzero-length prefix is applied to this filename, a
3325		slash shall be inserted between the prefix and the filename.
3326		A zero-length prefix is an obsolescent feature that indicates
3327		the current working directory. It appears as two adjacent
3328		colons (::), as an initial colon preceding the rest of the list, or
3329		as a trailing colon following the rest of the list. A Strictly
3330		Conforming POSIX.2 Application shall use an actual path-
3331		name (such as ' . ') to represent the current working direc-
3332		tory in PATH . The list shall be searched from beginning to
3333		end, applying the filename to each prefix, until an execut-
3334		able file with the specified name and appropriate execution
3335		permissions is found. If the pathname being sought con-
3336		tains a slash, the search through the path prefixes shall not
3337		be performed. If the pathname begins with a slash, the
3338		specified path shall be resolved as described in 2.2.2.104. If
3339		PATH is unset or is set to null, the path search is
3340		implementation-defined.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

3341	SHELL	A pathname of the user's preferred command language interpreter. If this interpreter does not conform to the shell
3342		command language in Section 3, utilities may behave dif-
3343		ferently than described in this standard.
3344		
3345	TMPDIR	A pathname of a directory made available for programs that
3346		need a place to create temporary files.
3347	TERM	The terminal type for which output is to be prepared. This
3348		information is used by utilities and application programs
3349		wishing to exploit special capabilities specific to a terminal.
3350		The format and allowable values of this environment vari-
3351		able are unspecified.
3352	TZ	Time-zone information. The format is described in
3353		POSIX.1 {8} 8.1.1.
3354	The environment variables LANG , LC_ALL , LC_COLLATE , LC_CTYPE ,	
3355	LC_MESSAGES , LC_MONETARY , LC_NUMERIC , and LC_TIME (LC_*) provide	
3356	for the support of internationalized applications. The standard utilities shall	
3357	make use of these environment variables as described in this clause and the indi-	
3358	vidual Environment Variables subclauses for the utilities. If these variables	
3359	specify locale categories that are not based upon the same underlying code set,	
3360	the results are unspecified.	
3361	For utilities used in internationalized applications, if the LC_ALL is not set in the	
3362	environment or is set to the empty string, and if any of LC_* variables is not set	
3363	in the environment or is set to the empty string, the operational behavior of the	
3364	utility for the corresponding locale category shall be determined by the setting of	
3365	the LANG environment variable. If the LANG environment variable is not set or	
3366	is set to the empty string, the implementation-defined default locale shall be used.	
3367	If LANG (or any of the LC_* environment variables) contains the value "C", or	
3368	the value "POSIX", the POSIX Locale shall be selected and the standard utilities	
3369	shall behave in accordance with the rules in the 2.5.1 for the associated category.	
3370	If LANG (or any of the LC_* environment variables) begins with a slash, it shall	
3371	be interpreted as the pathname of a file that was created in the output format	
3372	used by the <code>localedef</code> utility; see 4.35.6.3. Referencing such a pathname shall	
3373	result in that locale being used for the category indicated.	
3374	If LANG (or any of the LC_* environment variables) contains one of a set of	
3375	implementation-defined values, the standard utilities shall behave in accordance	
3376	with the rules in a corresponding implementation-defined locale description for	
3377	the associated category.	
3378	If LANG (or any of the LC_* environment variables) contains a value that the	
3379	implementation does not recognize, the behavior is unspecified.	
3380	Additional criteria for determining a valid locale name are implementation	
3381	defined.	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

3382 **2.6.1 Environment Variables Rationale.** *(This subclause is not a part of P1003.2)*

3383 The standard is worded so that the specified variables *may* be provided to the
 3384 application. There is no way that the implementation can guarantee that a utility
 3385 will ever see an environment variable, as a parent process can change the
 3386 environment for its children. The `env -i` command in this standard and the
 3387 POSIX.1 {8} *exec* family both offer ways to remove any of these variables from the
 3388 environment.

3389 The language about locale implies that any utilities written in Standard C and
 3390 conforming to POSIX.2 must issue the following call:

```
3391     setlocale(LC_ALL, " ")
```

3392 If this were omitted, the C Standard {7} specifies that the C Locale would be used.

3393 If any of the environment variables is invalid, it makes sense to default to an
 3394 implementation-defined, consistent locale environment. It is more confusing for a
 3395 user to have partial settings occur in case of a mistake. All utilities would then
 3396 behave in one language/cultural environment. Furthermore, it provides a way of
 3397 forcing the whole environment to be the implementation-defined default. Disas-
 3398 trous results could occur if a pipeline of utilities partially use the environment
 3399 variables in different ways. In this case, it would be appropriate for utilities that
 3400 use **LANG** and related variables to exit with an error if any of the variables are
 3401 invalid. For example, users typing individual commands at a terminal might
 3402 want `date` to work if **LC_MONETARY** is invalid as long as **LC_TIME** is valid.
 3403 Since these are conflicting reasonable alternatives, POSIX.2 leaves the results
 3404 unspecified if the locale environment variables would not produce a complete
 3405 locale matching the user's specification.

3406 The locale settings of individual categories cannot be truly independent and still
 3407 guarantee correct results. For example, when collating two strings, characters
 3408 must first be extracted from each string (governed by **LC_CTYPE**) before being
 3409 mapped to collating elements (governed by **LC_COLLATE**) for comparison. That
 3410 is, if **LC_CTYPE** is causing parsing according to the rules of a large, multibyte
 3411 code set (potentially returning 20 000 or more distinct character code set values),
 3412 but **LC_COLLATE** is set to handle only an 8-bit code set with 256 distinct charac-
 3413 ters, meaningful results are obviously impossible.

3414 The **LC_MESSAGES** variable affects the language of messages generated by the
 3415 standard utilities. This standard does not provide a means whereby applications
 3416 can easily be written to perform similar feats. Future versions of POSIX.1 {8} and
 3417 POSIX.2 are expected to provide both functions and utilities to accomplish mul-
 3418 tilanguage messaging (using message catalogs), but such facilities were not ready
 3419 for standardization at the time the initial versions of the standards were
 3420 developed.

3421 This clause is not a full list of all environment variables, but only those of impor-
 3422 tance to multiple utilities. Nevertheless, to satisfy some members of the balloting
 3423 group, here is a list of the other environment variable symbols mentioned in this
 3424 standard:

	<u>Variable</u>	<u>Utility</u>	<u>Variable</u>	<u>Utility</u>
3425	CDPATH	cd	MAKEFLAGS	make
3426	COLUMNS	ls	OPTARG	getopts
3427	DEAD	mailx	OPTIND	getopts
3428	IFS	sh	PRINTER	lp
3429	LPDEST	lp	PS1	sh
3430	MAIL	sh	PS2	sh
3431	MAILRC	mailx		
3432				

1

3433 The description of **PATH** is similar to that in POSIX.1 {8}, except:

- 3434 — The behavior of a null prefix is marked obsolescent in favor of using a real
3435 pathname. This was done at the behest of some members of the balloting
3436 group, who apparently felt it offered a more secure environment, where the
3437 current directory would not be selected unintentionally.
- 3438 — The POSIX.1 {8} *exec* description requires an implementation-defined path
3439 search when **PATH** is “not present.” POSIX.2 spells out that this means
3440 “unset or set to null.” Many implementations historically have used a
3441 default value of /bin and /usr/bin. POSIX.2 does not mandate that this
3442 default path be identical to that retrieved from `getconf _CS_PATH`
3443 because it is likely that a transition to POSIX.2 conformance will see the
3444 newly-standardized utilities in another directory that needs to be isolated
3445 from some historical applications.
- 3446 — The POSIX.1 {8} **PATH** description is ambiguous about whether an “execut-
3447 able file” means one that has the appropriate permissions for the searching
3448 process to execute it. One reading would say that a file with any of the exe-
3449 cution bits set on would satisfy the search and that an [EACCES] could be
3450 returned at that point. This is not the way historical systems work and
3451 POSIX.2 has clarified it to mean that the path search will continue until it
3452 finds the name with the execute permissions that would allow the process
3453 to execute it. (The case of the [ENOEXEC] error is handled in the text of
3454 3.9.1.1.)

3455 The terminology “beginning to end” is used in **PATH** to avoid the noninternation-
3456 alized “left to right.” There is no way to have a colon character embedded within
3457 a pathname that is part of the **PATH** variable string. Colon is not a member of
3458 the portable filename character set, so this should not be a problem. A portable
3459 application can retrieve a default **PATH** value (that will allow access to all the
3460 standard utilities) from the system using the command:

3461 `getconf _CS_PATH`

3462 See the rationale with `command` for an example of using this.

3463 The **SHELL** variable names the user’s preferred shell; it is a guide to applications.
3464 There is no direct requirement that that shell conform to this standard—that
3465 decision should rest with the user. It is the intention of the developers of this
3466 standard that alternative shells be permitted, if the user chooses to develop or
3467 acquire one. An operating system that builds its shell into the “kernel” in such a
3468 manner that alternative shells would be impossible does not conform to the spirit

3469 of the standard.

3470 The following environment variables are not currently used by the standard utili-
3471 ties (although they may be by future UPE utilities). Implementations should
3472 reserve the names for the following purposes:

3473 **EDITOR** The name of the user's preferred text file editor. The value of this
3474 variable is the name of a utility: either a pathname containing a
3475 slash, or a filename to be located using the **PATH** environment
3476 variable.

3477 **VISUAL** The name of the user's preferred "visual," or full-screen, text file
3478 editor. The value of this variable is the name of a utility: either a
3479 pathname containing a slash, or a filename to be located using
3480 the **PATH** environment variable.

3481 The decision to restrict conforming systems to the use of digits, uppercase letters,
3482 and underscores for environment variable names allows applications to use lower-
3483 case letters in their environment variable names without conflicting with any con-
3484 forming system.

3485 **PROCLANG** was added to an earlier draft for internationalized applications, but
3486 was removed from the standard because the working group determined that it
3487 was not of use.

3488 **USER** was removed from an earlier draft because it was an unreasonable duplica-
3489 tion of **LOGNAME**.

2.7 Required Files

The following directories shall exist on conforming systems and shall be used as described. Strictly Conforming POSIX.2 Applications shall not assume the ability to create files in any of these directories.

/ The root directory.

/dev Contains /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 programs 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/sink. Data written to /dev/null is discarded. Reads from /dev/null 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.

2.7.1 Required Files Rationale. *(This subclause is not a part of P1003.2)*

A description of the historical /usr/tmp was omitted, removing any concept of differences in emphasis between the / and /usr versions. The descriptions of /bin, /usr/bin, /lib, and /usr/lib were omitted because they are not useful for applications. In an early draft, a distinction was made between *system* and *application* directory usage, but this was not found to be useful.

In Draft 8, /, /dev, /local, /usr/local, and /usr/man were removed. The directories / and /dev were restored in Draft 9. It was pointed out by several balloters that the notion of a hierarchical directory structure is key to other information presented in later sections of the standard. (Previously, some had argued that special devices and temporary files could conceivably be handled without a directory structure on some implementations. For example, the system could treat the characters “/tmp” as a special token that would store files using some non-POSIX file system structure. This notion was rejected by the working group, which requires that all the files in this clause be implemented via POSIX file systems.)

The /tmp directory is retained in the standard to accommodate historical applications that assume its availability. Future implementations are encouraged to provide suitable directory names in **TMPDIR** and future applications are encouraged

to use the contents of **TMPDIR** for creating temporary files.

The standard files `/dev/null` and `/dev/tty` are required to be both readable and writable to allow applications to have the intended historical access to these files.

2.8 Regular Expression Notation

Editor's Note: The entire rationale for this clause appears at the end of the clause.

Regular Expressions (REs) provide a mechanism to select specific strings from a set of character strings.

Regular expressions are a context-independent syntax that can represent a wide variety of character sets and character set orderings, where these character sets are interpreted according to the current locale. While many regular expressions can be interpreted differently depending on the current locale, many features, such as character class expressions, provide for contextual invariance across locales.

The Basic Regular Expression (BRE) notation and construction rules in 2.8.3 shall apply to most utilities supporting regular expressions. Some utilities, instead, support the Extended Regular Expressions (ERE) described in 2.8.4; any exceptions for both cases are noted in the descriptions of the specific utilities using regular expressions. Both BREs and EREs are supported by the Regular Expression Matching interface in 7.3.

2.8.1 Regular Expression Definitions

For the purposes of this clause, the following definitions apply.

2.8.1.1 entire regular expression: The concatenated set of one or more BREs or EREs that make up the pattern specified for string selection.

2.8.1.2 matched: A sequence of zero or more characters is said to be matched by a BRE or ERE when the characters in the sequence corresponds to a sequence of characters defined by the pattern.

Matching shall be based on the bit pattern used for encoding the character, not on the graphic representation of the character. 1

The search for a matching sequence shall start at the beginning of a string and stop when the first sequence matching the expression is found, where “first” is defined to mean “begins earliest in the string.” If the pattern permits a variable number of matching characters and thus there is more than one such sequence starting at that point, the longest such sequence shall be matched. For example: 1
the BRE `bb*` matches the second through fourth characters of `abbbbc`, and the ERE 1
`(wee|week)(knights|night)` matches all ten characters of `weeknights`. 1

Consistent with the whole match being the longest of the leftmost matches, each subpattern, from left to right, shall match the longest possible string. For this purpose, a null string shall be considered to be longer than no match at all. For example, matching the BRE `\(.*\).*` against `abcdef`, the subexpression `(\1)` is `abcdef`, and matching the BRE `\(a*\)` against `bc`, the subexpression `(\1)` is the null string.

When a multicharacter collating element in a bracket expression (see 2.8.3.2) is involved, the longest sequence shall be measured in characters consumed from the string to be matched; i.e., the collating element counts not as one element, but as the number of characters it matches.

2.8.1.3 BRE [ERE] matching a single character: A BRE or ERE that matches either a single character or a single collating element.

Only a BRE or ERE of this type that includes a bracket expression (see 2.8.3.2) can match a collating element.

2.8.1.4 BRE [ERE] matching multiple characters: A BRE or ERE that matches a concatenation of single characters or collating elements.

Such a BRE or ERE is made up from a *BRE (ERE) matching a single character* and *BRE (ERE) special characters*.

2.8.2 Regular Expression General Requirements

The requirements in this subclause shall apply to both basic and extended regular expressions.

The use of regular expressions is generally associated with text processing; i.e., REs (BREs and EREs) operate on text strings; i.e., zero or more characters followed by an end-of-string delimiter (typically NUL). Some utilities employing regular expressions limit the processing to lines; i.e., zero or more characters followed by a `<newline>`. In the regular expression processing described in this standard, the `<newline>` character is regarded as an ordinary character. This standard specifies within the individual descriptions of those standard utilities employing regular expressions whether they permit matching of `<newline>`s; if not stated otherwise, the use of literal `<newline>`s or any escape sequence equivalent produces undefined results.

The interfaces specified in this standard do not permit the inclusion of a NUL character in an RE or in the string to be matched. If during the operation of a standard utility a NUL is included in the text designated to be matched, that NUL may designate the end of the text string for the purposes of matching.

When a standard utility or function that uses regular expressions specifies that pattern matching shall be performed without regard to the case (upper- or lower-) of either data or patterns, then when each character in the string is matched against the pattern, not only the character, but also its case counterpart (if any), shall be matched.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

3606 The implementation shall support any regular expression that does not exceed
 3607 256 bytes in length.

3608 This clause uses the term “invalid” for certain constructs or conditions. Invalid 1
 3609 REs shall cause the utility or function using the RE to generate an error condition. 1
 3610 When “invalid” is not used, violations of the specified syntax or semantics for REs 1
 3611 produce undefined results: this may entail an error, enabling an extended syntax 1
 3612 for that RE, or using the construct in error as literal characters to be matched. 1

3613 2.8.3 Basic Regular Expressions

3614 2.8.3.1 BREs Matching a Single Character or Collating Element

3615 A BRE ordinary character, a special character preceded by a backslash, or a period
 3616 shall match a single character. A bracket expression shall match a single charac-
 3617 ter or a single collating element.

3618 2.8.3.1.1 BRE Ordinary Characters

3619 An ordinary character is a BRE that matches itself: any character in the sup-
 3620 ported character set, except for the BRE special characters listed in 2.8.3.1.2.

3621 The interpretation of an ordinary character preceded by a backslash (\) is
 3622 undefined, except for:

- 3623 (1) The characters), (, {, and }.
- 3624 (2) The digits 1 through 9 (see 2.8.3.3).
- 3625 (3) A character inside a bracket expression.

3626 2.8.3.1.2 BRE Special Characters

3627 A *BRE special character* has special properties in certain contexts. Outside of 1
 3628 those contexts, or when preceded by a backslash, such a character shall be a BRE 1
 3629 that matches the special character itself. The BRE special characters and the con-
 3630 texts in which they have their special meaning are:

- 3631 . [\ The period, left-bracket, and backslash shall be special except when
 3632 used in a bracket expression (see 2.8.3.2). An expression containing
 3633 a [that is not preceded by a backslash and is not part of a bracket
 3634 expression produces undefined results. 1
- 3635 * The asterisk is special except when used
- 3636 — In a bracket expression, 1
- 3637 — As the first character of an entire BRE (after an initial ^, if any), 1
 3638 or 1
- 3639 — As the first character of a subexpression (after an initial ^, if 1
 3640 any); see 2.8.3.3. 1

- 3641 [^] The circumflex shall be special when used 1
- 3642 — As an anchor (see 2.8.3.5) or, 1
- 3643 — As the first character of a bracket expression (see 2.8.3.2). 1
- 3644 \$ The dollar-sign shall be special when used as an anchor. 1

3645 2.8.3.1.3 Periods in BREs

- 3646 A period (.), when used outside of a bracket expression, is a BRE that shall match 1
- 3647 any character in the supported character set except NUL. 1

3648 2.8.3.2 RE Bracket Expression

- 3649 A bracket expression (an expression enclosed in square brackets, []) is an RE that 1
- 3650 matches a single collating element contained in the nonempty set of collating ele- 1
- 3651 ments represented by the bracket expression. 1

3652 The following rules and definitions apply to bracket expressions:

- 3653 (1) A *bracket expression* is either a matching list expression or a nonmatch- 1
- 3654 ing list expression. It consists of one or more expressions: collating ele- 1
- 3655 ments, collating symbols, equivalence classes, character classes, or range 1
- 3656 expressions. Strictly Conforming POSIX.2 Applications shall not use 1
- 3657 range expressions, but conforming implementations shall support regular 1
- 3658 expressions containing range expressions. The right-bracket (]) shall 1
- 3659 lose its special meaning and represent itself in a bracket expression if it 1
- 3660 occurs first in the list [after an initial circumflex (^), if any]. Otherwise, 1
- 3661 it shall terminate the bracket expression, unless it appears in a collating 1
- 3662 symbol (such as [.] .]) or is the ending right-bracket for a collating sym- 1
- 3663 bol, equivalence class, or character class). The special characters

3664 . * [\

- 3665 (period, asterisk, left-bracket, and backslash, respectively) shall lose 1
- 3666 their special meaning within a bracket expression. 1

3667 The character sequences

3668 [. [= [:

- 3669 (left-bracket followed by a period, equals-sign, or colon) shall be special 1
- 3670 inside a bracket expression and are used to delimit collating symbols, 1
- 3671 equivalence class expressions, and character class expressions. These 1
- 3672 symbols shall be followed by a valid expression and the matching ter- 1
- 3673 minating sequence .], =], or :], as described in the following items. 1

- 3674 (2) A *matching list* expression specifies a list that shall match any one of the 1
- 3675 expressions represented in the list. The first character in the list shall 1
- 3676 not be the circumflex. For example, [abc] is an RE that matches any of 1
- 3677 a, b, or c. 1
- 3678 (3) A *nonmatching list* expression begins with a circumflex (^), and specifies 1
- 3679 a list that shall match any character or collating element except for the 1

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

- expressions represented in the list after the leading circumflex. For example, `[^abc]` is an RE that matches any character or collating element except a, b, or c. The circumflex shall have this special meaning only when it occurs first in the list, immediately following the left-bracket.
- (4) A *collating symbol* is a collating element enclosed within bracket-period (`[. .]`) delimiters. Collating elements are defined as described in 2.5.2.2.4. Multicharacter collating elements shall be represented as collating symbols when it is necessary to distinguish them from a list of the individual characters that make up the multicharacter collating element. For example, if the string `ch` is a collating element in the current collation sequence with the associated collating symbol `<ch>`, the expression `[.ch.]` shall be treated as an RE matching the character sequence `ch`, while `[ch]` shall be treated as an RE matching `c` or `h`. Collating symbols shall be recognized only inside bracket expressions. This implies that the RE `[.ch.]*c` shall match the first through fifth character in the string `chchch`. If the string is not a collating element in the current collating sequence definition, or if the collating element has no characters associated with it (e.g., see the symbol `<HIGH>` in the example collation definition shown in 2.5.2.2.4), the symbol shall be treated as an invalid expression.
- (5) An *equivalence class expression* shall represent the set of collating elements belonging to an equivalence class, as described in 2.5.2.2.4. 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 (`[= =]`) delimiters. For example, if `a`, `à`, and `â` belong to the same equivalence class, then `[=a=]b`, `[=à=]b`, and `[=â=]b` shall each be 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 set of characters belonging to a character class, as defined in the `LC_CTYPE` category in the current locale. All character classes specified in the current locale shall be recognized. A character class expression shall be expressed as a character class name enclosed within “bracket-colon” (`[: :]`) delimiters.
- Strictly conforming POSIX.2 applications shall only use the following character class expressions, which shall be supported on all conforming implementations:
- | | | | |
|--------------------------|--------------------------|--------------------------|---------------------------|
| <code>[:alnum:]</code> | <code>[:cntrl:]</code> | <code>[:lower:]</code> | <code>[:space:]</code> |
| <code>[:alpha:]</code> | <code>[:digit:]</code> | <code>[:print:]</code> | <code>[:upper:]</code> |
| <code>[:blank:]</code> | <code>[:graph:]</code> | <code>[:punct:]</code> | <code>[:xdigit:]</code> |
- (7) A *range expression* represents the set of collating elements that fall between two elements in the current collation sequence, inclusively. It shall be expressed as the starting point and the ending point separated

3724 by a hyphen (-).

3725 Range expressions shall not be used in Strictly Conforming POSIX.2
 3726 Applications because their behavior is dependent on the collating
 3727 sequence. Range expressions shall be supported by conforming imple-
 3728 mentations.

3729 In the following, all examples assume the collation sequence specified for
 3730 the POSIX Locale, unless another collation sequence is specifically
 3731 defined.

3732 The starting range point and the ending range point shall be a collating
 3733 element or collating symbol. An equivalence class expression used as a 2
 3734 starting or ending point of a range expression produces unspecified 2
 3735 results. The ending range point shall collate equal to or higher than the 2
 3736 starting range point; otherwise the expression shall be treated as invalid.
 3737 The order used is the order in which the collating elements are specified
 3738 in the current collation definition. One-to-many mappings (see 2.5.2.2)
 3739 shall not be performed. For example, assuming that the character eszet
 3740 (ß) is placed in the basic collation sequence after r and s, but before t,
 3741 and that it maps to the sequence ss for collation purposes, then the
 3742 expression [r-s] matches only r and s, but the expression [s-t]
 3743 matches s, ß, or t.

3744 The interpretation of range expressions where the ending range point
 3745 also is the starting range point of a subsequent range expression is
 3746 undefined.

3747 The hyphen character shall be treated as itself if it occurs first (after an
 3748 initial ^, if any) or last in the list, or as an ending range point in a range
 3749 expression. As examples, the expressions [-ac] and [ac-] are
 3750 equivalent and match any of the characters a, c, or -; the expressions
 3751 [^ac] and [^ac-] are equivalent and match any characters except a, 1
 3752 c, or -; the expression [%--] matches any of the characters between % 1
 3753 and - inclusive; the expression [--@] matches any of the characters
 3754 between - and @, inclusive; and the expression [a--@] is invalid,
 3755 because the letter a follows the symbol - in the POSIX Locale. To use a
 3756 hyphen as the starting range point, it shall either come first in the
 3757 bracket expression or be specified as a collating symbol. For example:
 3758 [[] .- .]-0], which matches either a right bracket or any character or 1
 3759 collating element that collates between hyphen and 0, inclusive. 1

3760 2.8.3.3 BREs Matching Multiple Characters

3761 The following rules can be used to construct BREs matching multiple characters
 3762 from BREs matching a single character:

- 3763 (1) The concatenation of BREs shall match the concatenation of the strings
 3764 matched by each component of the BRE. 1
- 3765 (2) A *subexpression* can be defined within a BRE by enclosing it between the
 3766 character pairs \(and \). Such a subexpression shall match whatever

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

3767	it would have matched without the <code>\(</code> and <code>\)</code> , except that anchoring	1
3768	within subexpressions is optional behavior; see 2.8.3.5. Subexpressions	1
3769	can be arbitrarily nested.	1
3770	(3) The <i>backreference</i> expression <code>\n</code> shall match the same (possibly empty)	1
3771	string of characters as was matched by a subexpression enclosed between	1
3772	<code>\(</code> and <code>\)</code> preceding the <code>\n</code> . The character <i>n</i> shall be a digit from 1	
3773	through 9, specifying the <i>n</i> -th subexpression [the one that begins with	
3774	the <i>n</i> -th <code>\(</code> and ends with the corresponding paired <code>\)</code>]. The expression	
3775	is invalid if less than <i>n</i> subexpressions precede the <code>\n</code> . For example, the	
3776	expression <code>^\(.*\)\1\$</code> matches a line consisting of two adjacent	
3777	appearances of the same string, and the expression <code>\(a\)*\1</code> fails to	2
3778	match a.	2
3779	(4) When a BRE matching a single character, a subexpression, or a	1
3780	backreference is followed by the special character asterisk (<code>*</code>), together	1
3781	with that asterisk it shall match what zero or more consecutive	2
3782	occurrences of the BRE would match. For example, <code>[ab]*</code> and <code>[ab][ab]</code>	2
3783	are equivalent when matching the string <code>ab</code> .	2
3784	(5) When a BRE matching a single character, a subexpression, or a	1
3785	backreference is followed by an <i>interval expression</i> of the format <code>\{m\}</code> ,	1
3786	<code>\{m,\}</code> , or <code>\{m,n\}</code> , together with that interval expression it shall	1
3787	match what repeated consecutive occurrences of the BRE would match.	2
3788	The values of <i>m</i> and <i>n</i> shall be decimal integers in the range $0 \leq m \leq n \leq$	1
3789	<code>{RE_DUP_MAX}</code> , where <i>m</i> specifies the exact or minimum number of	
3790	occurrences and <i>n</i> specifies the maximum number of occurrences. The	
3791	expression <code>\{m\}</code> shall match exactly <i>m</i> occurrences of the preceding	
3792	BRE, <code>\{m,\}</code> shall match at least <i>m</i> occurrences, and <code>\{m,n\}</code> shall	
3793	match any number of occurrences between <i>m</i> and <i>n</i> , inclusive.	1
3794	For example, in the string <code>abababcccccccd</code> the BRE <code>c\{3\}</code> is matched	
3795	by characters seven through nine, the BRE <code>\(ab\)\{4,\}</code> is not	
3796	matched at all, and the BRE <code>c\{1,3\}d</code> is matched by characters ten	
3797	through thirteen.	
3798	The behavior of multiple adjacent duplication symbols (<code>*</code> and intervals) produces	1
3799	undefined results.	1
3800	2.8.3.4 BRE Precedence	1
3801	The order of precedence shall be as shown in Table 2-12, from high to low.	1
3802	2.8.3.5 BRE Expression Anchoring	
3803	A BRE can be limited to matching strings that begin or end a line; this is called	1
3804	<i>anchoring</i> . The circumflex and dollar-sign special characters shall be considered	1
3805	BRE anchors in the following contexts:	1
3806	(1) A circumflex (<code>^</code>) shall be an anchor when used as the first character of an	1
3807	entire BRE. The implementation may treat circumflex as an anchor when	1

Table 2-12 – BRE Precedence

1

3808

3809

3810

collation-related bracket symbols [= =] [: :] [. .]

1

3811

escaped characters \<special character>

1

3812

bracket expression []

1

3813

subexpressions/backreferences \(\) \n

1

3814

single-character-BRE duplication * \{m,n\}

1

3815

concatenation

1

3816

anchoring ^ \$

1

3817

3818

used as the first character of a subexpression. The circumflex shall

1

3819

anchor the expression (or optionally subexpression) to the beginning of a

1

3820

string; only sequences starting at the first character of a string shall be

1

3821

matched by the BRE. For example, the BRE ^ab matches ab in the string

1

3822

abcdef, but fails to match in the string cdefab. The BRE \(^ab\) may

1

3823

match the former string. A portable BRE shall escape a leading

1

3824

circumflex in a subexpression to match a literal circumflex.

1

3825

(2) A dollar-sign (\$) shall be an anchor when used as the last character of an

1

3826

entire BRE. The implementation may treat a dollar-sign as an anchor

1

3827

when used as the last character of a subexpression. The dollar-sign shall

1

3828

anchor the expression (or optionally subexpression) to the end of the

1

3829

string being matched; the dollar-sign can be said to match the “end-of-

1

3830

string” following the last character.

1

3831

(3) A BRE anchored by both ^ and \$ shall match only an entire string. For

2

3832

example, the BRE ^abcdef\$ matches strings consisting only of abcdef.

1

3833

2.8.4 Extended Regular Expressions

3834

The *extended regular expression* (ERE) notation and construction rules shall apply

3835

to utilities defined as using extended regular expressions; any exceptions to the

3836

following rules are noted in the descriptions of the specific utilities using EREs.

3837

2.8.4.1 EREs Matching a Single Character or Collating Element

3838

An ERE ordinary character, a special character preceded by a backslash, or a

1

3839

period shall match a single character. A bracket expression shall match a single

1

3840

character or a single collating element. An *ERE matching a single character*

3841

enclosed in parentheses shall match the same as the ERE without parentheses

3842

would have matched.

2.8.4.1.1 ERE Ordinary Characters

An *ordinary character* is an ERE that matches itself. An ordinary character is any character in the supported character set, except for the ERE special characters listed in 2.8.4.1.2. The interpretation of an ordinary character preceded by a backslash (\) is undefined.

2.8.4.1.2 ERE Special Characters

An *ERE special character* has special properties in certain contexts. Outside of those contexts, or when preceded by a backslash, such a character shall be an ERE that matches the special character itself. The extended regular expression special characters and the contexts in which they shall have their special meaning are:

- . [\ (The period, left-bracket, backslash, and left-parenthesis are special except when used in a bracket expression (see 2.8.3.2).
- * + ? { The asterisk, plus-sign, question-mark, and left-brace are special except when used in a bracket expression (see 2.8.3.2). Any of the following uses produce undefined results:
 - If these characters appear first in an ERE, or immediately following a vertical-line, circumflex, or left-parenthesis.
 - If a left-brace is not part of a valid interval expression.
- | The vertical-line is special except when used in a bracket expression (see 2.8.3.2). A vertical-line appearing first or last in an ERE, or immediately following a vertical-line or a left-parentheses, produces undefined results.
- ^ The circumflex shall be special when used
 - As an anchor (see 2.8.4.6) or,
 - As the first character of a bracket expression (see 2.8.3.2).
- \$ The dollar-sign shall be special when used as an anchor.

2.8.4.1.3 Periods in EREs

A period (.), when used outside of a bracket expression, is an ERE that shall match any character in the supported character set except NUL.

2.8.4.2 ERE Bracket Expression

The rules for ERE Bracket Expressions are the same as for Basic Regular Expressions; see 2.8.3.2.

2.8.4.3 EREs Matching Multiple Characters

The following rules shall be used to construct EREs matching multiple characters from EREs matching a single character:

- (1) A *concatenation of EREs* shall match the concatenation of the character sequences matched by each component of the ERE. A concatenation of EREs enclosed in parentheses shall match whatever the concatenation without the parentheses matches. For example, both the ERE `cd` and the ERE `(cd)` are matched by the third and fourth character of the string `abcdefabcdef`.
- (2) When an ERE matching a single character, or a concatenation of EREs enclosed in parentheses is followed by the special character plus-sign (+), together with that plus-sign it shall match what one or more consecutive occurrences of the ERE would match. For example, the ERE `b+(bc)` matches the fourth through seventh characters in the string `acabbbbcde`. And, `[ab]+` and `[ab][ab]*` are equivalent.
- (3) When an ERE matching a single character, or a concatenation of EREs enclosed in parentheses is followed by the special character asterisk (*), together with that asterisk it shall match what zero or more consecutive occurrences of the ERE would match. For example, the ERE `b*c` matches the first character in the string `cabbbbcde`, and the ERE `b*cd` matches the third through seventh characters in the string `cabbbbcdebbbbbbcbdbc`. And, `[ab]*` and `[ab][ab]` are equivalent when matching the string `ab`.
- (4) When an ERE matching a single character, or a concatenation of EREs enclosed in parentheses is followed by the special character question-mark (?), together with that question-mark it shall match what zero or one consecutive occurrences of the ERE would match. For example, the ERE `b?c` matches the second character in the string `acabbbbcde`.
- (5) When an ERE matching a single character, or a concatenation of EREs enclosed in parentheses is followed by an *interval expression* of the format `{m}`, `{m,}`, or `{m,n}`, together with that interval expression it shall match what repeated consecutive occurrences of the ERE would match. The values of *m* and *n* shall be decimal integers in the range $0 \leq m \leq n \leq \{\text{RE_DUP_MAX}\}$, where *m* specifies the exact or minimum number of occurrences and *n* specifies the maximum number of occurrences. The expression `{m}` shall match exactly *m* occurrences of the preceding ERE, `{m,}` shall match at least *m* occurrences, and `{m,n}` shall match any number of occurrences between *m* and *n*, inclusive.

For example, in the string `abababcccccd` the ERE `c{3}` is matched by characters seven through nine, and the ERE `(ab){2,}` is matched by characters one through six.

The behavior of multiple adjacent duplication symbols (+, *, ?, and intervals) produces undefined results.

2.8.4.4 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.

2.8.4.5 ERE Precedence

The order of precedence shall be as shown in Table 2-13, from high to low.

Table 2-13 – ERE Precedence

<i>collation-related bracket symbols</i>	<code>[= =] [: :] [. .]</code>	
<i>escaped characters</i>	<code>\<special character></code>	
<i>bracket expression</i>	<code>[]</code>	
<i>grouping</i>	<code>()</code>	
<i>single-character-ERE duplication</i>	<code>* + ? {m,n}</code>	
<i>concatenation</i>		
<i>anchoring</i>	<code>^ \$</code>	
<i>alternation</i>	<code> </code>	

For example, the ERE `abba|cde` matches either the string `abba` or the string `cde` (because concatenation has a higher order of precedence than alternation).

2.8.4.6 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 in the following contexts:

- (1) A circumflex (^) shall be an anchor when used anywhere outside a bracket expression. The circumflex shall anchor the (sub)expression to the beginning of a string; only sequences starting at the first character of a string shall be matched by the ERE. For example, the EREs `^ab` and `(^ab)` match `ab` in the string `abcdef`, but fail to match in the string `cdefab`.
- (2) A dollar-sign (\$) shall be an anchor when used anywhere outside a bracket expression. It shall anchor the expression to the end of the string being matched; the dollar-sign can be said to match the “end-of-string” following the last character.
- (3) An ERE anchored by both ^ and \$ shall match only an entire string. For example, the EREs `^abcdef$` and `(^abcdef$)` match strings consisting only of `abcdef`.

2.8.5 Regular Expression Grammar

Grammars describing the syntax of both basic and extended regular expressions are presented in this subclause. See the grammar conventions in 2.1.2.

2.8.5.1 BRE/ERE Grammar Lexical Conventions

The lexical conventions for regular expressions shall be as described in this subclause.

Except as noted, the longest possible token or delimiter beginning at a given point shall be recognized.

The following tokens shall be processed (in addition to those string constants shown in the grammar):

COLL_ELEM	Shall be any single-character collating element, unless it is a META_CHAR.	
BACKREF	(Applicable only to basic regular expressions.) Shall be the character string consisting of '\ ' followed by a single-digit numeral, 1 through 9.	1
DUP_COUNT	Shall represent a numeric constant. It shall be an integer in the range $0 \leq \text{DUP_COUNT} \leq \{\text{RE_DUP_MAX}\}$. This token shall only be recognized when the context of the grammar requires it. At all other times, digits not preceded by '\ ' shall be treated as ORD_CHAR.	1
META_CHAR	Shall be one of the characters:	
	^ When found first in a bracket expression	
	- When found anywhere but first (after an initial ^, if any) or last in a bracket expression, or as the ending range point in a range expression	
] When found anywhere but first (after an initial ^, if any) in a bracket expression.	
L_ANCHOR	(Applicable only to basic regular expressions.) Shall be the character ^ when it appears as the first character of a basic regular expression and when not QUOTED_CHAR. The ^ may be recognized as an anchor elsewhere; see 2.8.3.5.	1
ORD_CHAR	Shall be a character, other than one of the special characters in SPEC_CHAR.	1
QUOTED_CHAR	Shall be one of the character sequences:	1
	\^ \. * \[\\$ \\	1
R_ANCHOR	(Applicable only to basic regular expressions). Shall be the character \$ when it appears as the last character of a basic regular expression and when not QUOTED_CHAR. The \$ may	1

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

3994		be recognized as an anchor elsewhere; see 2.8.3.5.	1
3995	SPEC_CHAR	For basic regular expressions, shall be one of the following	
3996		special characters:	
3997	.	Anywhere outside bracket expressions	
3998	\	Anywhere outside bracket expressions	
3999	[Anywhere outside bracket expressions	
4000	^	When an anchor; see 2.8.3.5	2
4001	\$	When an anchor; see 2.8.3.5	2
4002	*	Anywhere except: first in an entire RE; anywhere	
4003		in a bracket expression; directly following \ (;	
4004		directly following an anchoring ^.	
4005		For extended regular expressions, shall be one of the follow-	
4006		ing special characters found anywhere outside bracket	
4007		expressions:	
4008	^ . [\$ () * + ? { \		
4009		(The close-parenthesis shall be considered special in this con-	2
4010		text only if matched with a preceding open-parenthesis.)	2

4011 2.8.5.2 RE and Bracket Expression Grammar

4012 This subclause presents the grammar for basic regular expressions, including the
4013 bracket expression grammar that is common to both BREs and EREs.

4014	%token	ORD_CHAR QUOTED_CHAR SPEC_CHAR DUP_COUNT	
4015	%token	BACKREF L_ANCHOR R_ANCHOR	
4016	%token	Back_open_paren Back_close_paren	
4017	/*	'\(' '\)' */	
4018	%token	Back_open_brace Back_close_brace	
4019	/*	'\{' '\}' */	
4020	/*	The following tokens are for the Bracket Expression	
4021		grammar common to both REs and EREs. */	
4022	%token	COLL_ELEM META_CHAR	1
4023	%token	Open_equal Equal_close Open_dot Dot_close Open_colon Colon_close	1
4024	/*	'[' '=' '[' '.' '.' '[:' ':']' */	1
4025	%token	class_name	
4026	/*	class_name is a keyword to the LC_CTYPE locale category */	
4027	/*	(representing a character class) in the current locale */	
4028	/*	and is only recognized between [: and :] */	
4029	%start	basic_reg_exp	

```

4030  %%

4031  /*
4032      Basic Regular Expression
4033  */

4034  */

4035  basic_reg_exp      :      RE_expression
4036                      | L_ANCHOR
4037                      |      R_ANCHOR
4038                      | L_ANCHOR      R_ANCHOR
4039                      | L_ANCHOR RE_expression
4040                      |      RE_expression R_ANCHOR
4041                      | L_ANCHOR RE_expression R_ANCHOR
4042                      ;

4043  RE_expression      :      simple_RE
4044                      | RE_expression simple_RE
4045                      ;

4046  simple_RE          : nondupl_RE
4047                      | nondupl_RE RE_dupl_symbol
4048                      ;

4049  nondupl_RE         : one_character_RE
4050                      | Back_open_paren RE_expression Back_close_paren
4051                      | Back_open_paren Back_close_paren
4052                      | BACKREF
4053                      ;

4054  /*
4055      Note: This grammar does not permit L_ANCHOR or
4056      R_ANCHOR inside \ ( and \ ) (which implies that ^ and $
4057      are ordinary characters). This reflects the semantic
4058      limits on the application, as noted in 2.8.3.5.
4059      Implementations are permitted to extend the language to
4060      interpret ^ and $ as anchors in these locations, and as
4061      such portable applications shall not use unescaped ^
4062      and $ in positions inside \ ( and \ ) that might be
4063      interpreted as anchors.
4064  */

4065  one_character_RE    : ORD_CHAR
4066                      | QUOTED_CHAR
4067                      | '.'
4068                      | bracket_expression
4069                      ;

4070  RE_dupl_symbol      : '*'
4071                      | Back_open_brace DUP_COUNT      Back_close_brace
4072                      | Back_open_brace DUP_COUNT ' , '      Back_close_brace
4073                      | Back_open_brace DUP_COUNT ' , ' DUP_COUNT Back_close_brace
4074                      ;

4075  /*
4076      Bracket Expression
4077  */
4078  */

```

```

4079 bracket_expression : '[' matching_list      ']'
4080                       | '[' nonmatching_list ']'
4081                       ;
4082 matching_list       : bracket_list
4083                       ;
4084 nonmatching_list    : '^' bracket_list
4085                       ;
4086 bracket_list        : follow_list
4087                       | follow_list '-'
4088                       ;
4089 follow_list         :          expression_term
4090                       | follow_list expression_term
4091                       ;
4092 expression_term     : single_expression
4093                       | range_expression
4094                       ;
4095 single_expression   : end_range
4096                       | character_class
4097                       ;
4098 range_expression    : start_range end_range
4099                       | start_range '-'
4100                       ;
4101 start_range         : end_range '-'
4102                       ;
4103 end_range           : COLL_ELEM
4104                       | collating_symbol
4105                       ;
4106 collating_symbol    : Open_dot COLL_ELEM Dot_close
4107                       | Open_dot META_CHAR Dot_close
4108                       ;
4109 equivalence_class   : Open_equal COLL_ELEM Equal_close
4110                       ;
4111 character_class     : Open_colon class_name Colon_close
4112                       ;

```

4113 2.8.5.3 ERE Grammar

4114 This subclause presents the grammar for extended regular expressions, excluding
4115 the bracket expression grammar.

4116 NOTE: The bracket expression grammar and the associated %token lines are identical between
4117 BREs and EREs. It has been omitted from the ERE subclause to avoid unnecessary editorial duplica-
4118 tion.

```

4119 %token  ORD_CHAR QUOTED_CHAR SPEC_CHAR DUP_COUNT
4120 %start  extended_reg_exp

```

```

4121  %%
4122  /*          -----
4123             Extended Regular Expression
4124             -----
4125  */
4126  extended_reg_exp :          anchored_ERE
4127                  |          nonanchored_ERE
4128                  | extended_reg_exp '|' nonanchored_ERE
4129                  | extended_reg_exp '|' anchored_ERE
4130                  ;
4131  anchored_ERE    : '^' nonanchored_ERE
4132                  | '^' nonanchored_ERE '$'
4133                  |      nonanchored_ERE '$'
4134                  | '^'
4135                  |          '$'
4136                  | '^'          '$'
4137                  ;
4138  nonanchored_ERE :          ERE_expression
4139                  | nonanchored_ERE ERE_expression
4140                  ;
4141  ERE_expression  : one_character_ERE
4142                  | '(' extended_reg_exp ')'
4143                  | ERE_expression ERE_dupl_symbol
4144                  ;
4145  one_character_ERE : ORD_CHAR
4146                  | '\' SPEC_CHAR
4147                  | '.'
4148                  | bracket_expression
4149                  ;
4150  ERE_dupl_symbol  : '*'
4151                  | '+'
4152                  | '?'
4153                  | '{' DUP_COUNT          '}'
4154                  | '{' DUP_COUNT ','      '}'
4155                  | '{' DUP_COUNT ',' DUP_COUNT '}'
4156                  ;

```

2.8.6 Regular Expression Notation Rationale. *(This subclause is not a part of P1003.2)*

Editor's Note: Some of the text and headings of this rationale have been rearranged. Moved text has not been diffmarked unless it changed.

Rather than repeating the description of regular expressions for each utility supporting REs, the working group preferred a common, comprehensive description of regular expressions in one place. The most common behavior is described here, and exceptions or extensions to this are documented for the respective utilities, if appropriate.

The Basic Regular Expression corresponds to the `ed` or historical `grep` type, and the Extended Regular Expression corresponds to the historical `egrep` type (now

4168 `grep -E`).

4169 The text is based on the `ed` description and substantially modified, primarily to
 4170 aid developers and others in the understanding of the capabilities and limitations
 4171 of regular expressions. Much of this was influenced by the internationalization
 4172 requirements.

4173 It should be noted that the definitions in this clause do not cover the `tr` utility
 4174 (see 4.64); the `tr` syntax does not employ regular expressions.

4175 The specification of regular expressions are particularly important to internation-
 4176 alization, because pattern matching operations are very basic operations in busi-
 4177 ness and other operations. The syntax and rules of regular expressions are
 4178 intended to be as intuitive as possible, to make them easy to understand and use.
 4179 The historical rules and behavior do not provide that capability to non-English-
 4180 language users, and does not provide the necessary support for commonly used
 4181 characters and language constructs. It was necessary to provide extensions to the
 4182 historical regular expression syntax and rules, to accommodate other languages.
 4183 Such modifications were proposed by the UniForum Technical Committee Sub-
 4184 committee on Internationalization and accepted by the working group. As they
 4185 are limited to bracket expressions, the rationale for these modifications can be
 4186 found in 2.8.6.3.2.

4187 **2.8.6.1 Regular Expression Definitions Rationale.** *(This subclause is not a part of* 4188 *P1003.2)*

4189 The definition of which sequence is matched when several are possible is based on
 4190 the leftmost-longest rule historically used by deterministic recognizers. This rule 1
 4191 is much easier to define and describe, and arguably more useful, than the first-
 4192 match rule historically used by nondeterministic recognizers. It is thought that
 4193 dependencies on the choice of rule are rare; carefully-contrived examples are
 4194 needed to demonstrate the difference.

4195 A formal expression of the leftmost-longest rule is: 1

4196 The search is performed as if all possible suffixes of the string were
 4197 tested for a prefix matching the pattern; the longest suffix containing
 4198 a matching prefix is chosen, and the longest possible matching prefix
 4199 of the chosen suffix is identified as the matching sequence.

4200 It is possible to determine what strings correspond to subexpressions by recur- 1
 4201 sively applying the leftmost longest rule to each subexpression, but only with the 1
 4202 proviso that the overall match is leftmost longest (see 2.8.1.2). For example, 1
 4203 matching `\(ac*\)c*d[ac]*\1` against `acdacaaa` should match `acdacaaa` (with 1
 4204 `\1=a`); simply matching the longest match for `\(ac*\)` would yield `\1=ac`, but 1
 4205 the overall match would be smaller (`acdac`). In principle, the implementation 1
 4206 must examine every possible match and among those that yield the leftmost long- 1
 4207 est total matches, pick the one that does the longest match for the leftmost subex- 1
 4208 pression and so on. Note that this means that matching by subexpressions is con- 1
 4209 text dependent: a subexpression within a larger RE may match a different string 1
 4210 from the one it would match as an independent RE, and two instances of the same 1

subexpression within the same larger RE may match different lengths even in similar sequences of characters. For example, in the ERE `(a.*b)(a.*b)`, the two identical subexpressions would match four and six characters, respectively, of `accbaccbb`. Thus, it is not possible to hierarchically decompose the matching problem into smaller, independent, matching problems.

Matching is based on the bit pattern used for encoding the character, not on the graphic representation of the character. This means that if a character set contains two or more encodings for a graphic symbol, or if the strings searched contain text encoded in more than one code set, no attempt is made to search for any other representation of the encoded symbol. If that is required, the user can specify equivalence classes containing all variations of the desired graphic symbol.

The definition of “single character” has been expanded to include also collating elements consisting of two or more characters; this expansion is applicable only when a bracket expression is included in the BRE or ERE. An example of such a collating element may be the Dutch “ij”, which collates as a “y.” In some encodings, a ligature “i with j” exists *as a character*, and would represent a single-character collating element. In another encoding, no such ligature exists, and the two-character sequence “ij” is defined as a multicharacter collating element. Outside brackets, the “ij” is treated as a two-character RE and will match the same characters in a string. Historically, a bracket expression only matched a single character. If, however, the bracket expression defines, for example, a range that includes “ij”, then this particular bracket expression will also match a sequence of the two characters “i” and “j” in the string.

2.8.6.2 Regular Expression General Requirements Rationale. *(This subclause is not a part of P1003.2)*

Historically, most regular expression implementations only match lines, not strings. However, that is more an effect of the usage than of an inherent feature of regular expressions itself. Consequently, POSIX.2 does not regard `<newline>s` as special; they are ordinary characters, and both a period and a nonmatching list can match them. Those utilities (like `grep`) that do not allow `<newline>s` to match are responsible for eliminating any `<newline>` from strings before matching against the RE. The `regcomp()` function, however, can provide support for such processing without violating the rules of this clause.

The definition of case-insensitive processing is intended to allow matching of multicharacter collating elements as well as characters. For instance, as each character in the string is matched using both its cases, the RE `[[.Ch.]]`, when matched against `char`, is in reality matched against `ch`, `Ch`, `cH`, and `CH`.

Some implementations of `egrep` have had very limited flexibility in handling complex extended regular expressions. POSIX.2 does not attempt to define the complexity of a BRE or ERE, but does place a lower limit on it—any regular expression must be handled, as long as it can be expressed in 256 bytes or less. (Of course, this does not place an upper limit on the implementation.) There are existing programs using a nondeterministic-recognizer implementation that

should have no difficulty with this limit. It is possible that a good approach would be to attempt to use the faster, but more limited, deterministic recognizer for simple expressions and to fall back on the nondeterministic recognizer for those expressions requiring it. Nondeterministic implementations must be careful to observe the 2.8.1.2 rules on which match is chosen; the longest match, not the first match, starting at a given character is used.

The term “invalid” highlights a difference between this clause and some others: POSIX.2 frequently avoids mandating of errors for syntax violations because they can be used by implementors to trigger extensions. However, the authors of the internationalization features of regular expressions desired to mandate errors for certain conditions to identify usage problems or nonportable constructs. These are identified within this rationale as appropriate. The remaining syntax violations have been left implicitly or explicitly undefined. For example, the BRE construct `\{1,2,3\}` does not comply with the grammar. A conforming application cannot rely on it producing an error nor matching the literal characters `\{1,2,3\}`. The term “undefined” was used in favor of “unspecified” because many of the situations are considered errors on some implementations and it was felt that consistency throughout the clause was preferable to mixing undefined and unspecified.

2.8.6.3 Basic Regular Expressions Rationale. *(This subclause is not a part of P1003.2)*

2.8.6.3.1 BREs Matching a Single Character or Collating Element Rationale. *(This subclause is not a part of P1003.2)*

2.8.6.3.2 RE Bracket Expression Rationale. *(This subclause is not a part of P1003.2)*

If a bracket expression must specify both `-` and `]`, then the `]` must be placed first (after the `^`, if any) and the `-` last within the bracket expression.

Range expressions are, historically, an integral part of regular expressions. However, the requirements of “natural language behavior” and portability does conflict: ranges must be treated according to the current collating sequence, and include such characters that fall within the range based on that collating sequence, regardless of character values. This, however, means that the interpretation will differ depending on collating sequence. If, for instance, one collating sequence defines “ä” as a variant of “a”, while another defines it as a letter following “z”, then the expression `[ä-z]` is valid in the first language and invalid in the second. This kind of ambiguity should be avoided in portable applications, and therefore the working group elected to state that ranges must not be used in strictly conforming applications; however, implementations must support them.

Some historical implementations allow range expressions where the ending range point of one range is also the starting point of the next (for instance `[a-m-o]`). This behavior should not be permitted, but to avoid breaking existing implementations, it is now *undefined* whether it is a valid expression, and how it should be interpreted.

Current practice in `awk` and `lex` is to accept escape sequences in bracket expressions as per Table 2-15, while the normal regular expression behavior is to regard such a sequence as consisting of two characters. Allowing the `awk/lex` behavior in regular expressions would change the normal behavior in an unacceptable way; it is expected that `awk` and `lex` will decode escape sequences in regular expressions before passing them to `regcomp()` or comparable routines. Each utility describes the escape sequences it accepts as an exception to the rules in this clause; the list is not the same, for historical reasons.

As noted earlier, the new syntax and rules have been added to accommodate other languages than English. These modifications were proposed by the UniForum Subcommittee on Internationalization and accepted by the working group. The remainder of this clause describes the rationale for these modifications.

Internationalization Requirements

The goal of the internationalization effort was to provide functions and capabilities that matched the capabilities of existing implementations, but that adhered to the user's local customs, rules, and environment. This has also been described as "removing the ASCII (and English language) bias."

In addition, other requirements also influence the standardization efforts, such as *portability*, *extensibility*, and *compatibility*.

In a worldwide environment *portability* carries much weight. Wherever feasible, users should be given the capability to develop code that can execute independently of character set, code set, or language.

Standards must also be *extensible*; to support further development, to allow for local or regional extensions, or to accommodate new concepts (such as multibyte characters).

Compatibility does not only refer to support of existing code, but also to making the new syntax, semantics, and functions compatible with existing environments and implementations.

Internationalization Technical Background

The C Standard [7] (and, by implication, also POSIX) recognizes that the ASCII character set used in historical UNIX system implementations is not adequate outside the Anglo-American language area. It is, however, not enough to remove the ASCII bias; the dependency on Anglo-Saxon conventions and rules must also be broadened to accommodate other cultures, including those that require thousands of characters.

Character sets are defined by their *attributes*; typical attributes are the *encoding*, the *collating sequence*, the *character classification*, and the *case mapping*.

It is also recognized that, even within one language area, several combinations of attributes exist: character set attributes are *mutable* and *combinatory*. So, rather than replacing one straitjacket by another, the proposed standards make character sets *user-definable* and *program-selectable*.

The existence of character set attributes is implicit in regular expressions (REs). This implies that regular expressions must recognize and adapt to the *program-selected* set of attributes.

A program *selects* the appropriate character set (or combination of attributes) using the mechanism described in 2.5. The *definition* of a character set (its attributes) is *external* to an executing program. Many combinations of attributes can exist concurrently. Of particular interest are the following attributes:

- (1) *Collating Sequence*. In existing implementations, the *encoded* ASCII ordering matches the *logical* English collating sequence. This correspondence does not exist for all code sets or languages. In addition, many languages employ concepts that have no counterparts in English collation:

- (a) In many languages, ordering is based on the concept of *string collation* rather than *character collation* as in English. One of the effects of this is that the ordering is based on *collating elements* rather than on characters. Characters typically map into collating elements:

One-to-one mapping, where a character is also a collating element,

One-to-N mapping, where a single character maps into two or more collating elements (as the German “ß” (eszet), which collates as “ss”),

N-to-one mapping, where two or more characters map into one collating element (as in the Spanish “ll”, which collates between “l” and “m”; i.e., a word beginning with “ll” collates *after* a word beginning with “lo”).

- (b) A common method for adding characters to an alphabet is to use diacritical marks, such as accents or circumflex (‘^’). In some languages, this creates a completely new character, collated differently from the Latin “base.” In other languages these accented characters are collated as variants of the Latin base letter; i.e., they have the same relative order; they are *equivalent*.

If the strings (words) being compared are equal except for “accents,” the strings can be ordered based on a secondary ordering *within* the “equivalence class.” For instance, in French, the words “tache”, “tâche”, and “tacheter” collate in that order.

The C Standard [7] recognizes this; it includes new library functions capable of handling complex collation rules. These functions depend on the setting of the *setlocale()* category LC_COLLATE for a definition of the current collation rules.

- (2) *Character Classification*. Character classification and case mapping is another area where each language (or even language area) has its own rules. Although users in different countries can use the same code set,

such as ISO 8859-1 {5}, the definition of what constitutes a letter or an uppercase letter may vary.

The C Standard {7} recognizes this; library functions used to classify characters or perform case mapping depend on the *setlocale()* category LC_CTYPE for a definition of how characters map to character classes.

Internationalization Proposal Areas

Based on the requirements and attribute characteristics defined above, and after reviewing proposals and definitions by X/Open and other organizations, the Uni-Forum Subcommittee on Internationalization decided to concentrate on the following areas: the range expression, character classes, the definition of one-character RE (multicharacter element), and equivalence classes.

Most of these are heavily dependent on the current definition of collation sequence; the Subcommittee felt it natural to couple the capabilities and interpretation of bracket expressions closely to the requirements for extended collation capabilities.

In addition, the Subcommittee felt that the capabilities described in 2.5 formed a suitable basis for runtime control of regular expression behavior.

The Subcommittee realized that the mechanism selected requires changes in the existing syntax. As a rule, the Subcommittee wished to minimize changes and avoid syntactical changes that may cause existing regular expressions to fail.

- (1) *Collating Elements and Symbols.* As noted above, many expressions within a bracket expression are closely connected with collation, and the Subcommittee defined many capabilities in terms of collating elements and collating symbols.

A collating element is defined as a sequence of one or more bytes defined in the current collating sequence definition as a unit of collation. In most cases, a collating element is equal to a character, but the collation sequence may exclude some characters, or define two or more characters as a collating element.

A one-character RE is, logically enough, defined as one character or something that translates into one character (the number of bits used to represent the character is not an issue here). The expression within square brackets is a one-character RE; i.e., single characters are matched against the list of single characters defined within the brackets.

In Spanish, the phrase “a to d” means the sequence of collating elements a, á, b, c, ch, and d. Consequently, with a Spanish character set, the range statement [a–d] includes the ch collating element, even though it is expressed with two characters (N-to-1 mapping).

The historical syntax, however, does not allow the user to define either the range from a through ch, or to define ch as a single character rather than as either c or h.

The Subcommittee decided that N-to-1 mappings be recognized (if properly delimited), as *one-character REs* inside, but not outside, square brackets (e.g., a period will never match `ch`).

To be distinguishable from a list of the characters themselves, the multicharacter element must be delimited from the remainder of the characters in the string. The characters `[.` and `.]` are used to delimit a multicharacter collating element from other elements, and can be used to delimit single-character collating elements.

- (2) *Equivalence Classes*. As stated previously, many languages extend the Latin alphabet by using diacritical marks. In some cases, the Latin base character (e.g., `a`) and the accented versions of the base (e.g., `â`, `â` in French) constitute a “subclass” of characters with some partially equivalent characteristics but different code values. Because these characters are related, they are often processed as a group. The historical syntax, however, does not provide for this in a portable manner.

Although it represents an extension of the historical capabilities, the X/Open group strongly recommended that a properly delimited collating element be recognized as representing an equivalence class, that is as the collating element itself, and all other characters with the same primary order in the collation sequence.

The Subcommittee supported this recommendation, and also selected `[=` and `=]` as delimiters for equivalence classes.

- (3) *Range Expressions*. The hyphen historically indicated “a range of consecutive ASCII characters;” typically it stands for the word “to,” as in “a to z,” and implies an ordered *interval*. In ASCII, the *encoded* order matches the *logical* English order; this is not true with other encodings or with other alphabets.

If the ASCII dependency is removed, an alternative could have been to use the encoded sequence of whatever code set is currently used. This, however, would certainly decrease portability, as well as requiring the user to know the ordering of the current code set. It would also most certainly be counter-intuitive; a French user would expect the expression `[a-d]` to match any of the letters `a`, `â`, `â`, `b`, `c`, `ç` or `d`. The Subcommittee regards this interpretation of ranges as most compatible with existing capabilities, and one that provides for the desired portability.

As the *logical* ordering need not be inherent in the *encoded* sequence, an external definition was required. Such a definition was already present via the *collating sequence* attribute of the character set. The `setlocale()` function provides for an `LC_COLLATE` category, which defines the current collating sequence. The Subcommittee selected this as the basis for the interpretation of ranges, as well as of equivalence classes and multicharacter collating symbols.

- (4) *Character Classes*. The *range* expression is commonly used to indicate a *character class*; the *ex(au_cmd)* section of the *SVID* states: “... a pair of

characters separated by – defines a range (e.g., a–z defines any lowercase letter)... In reality, [a–z] means “any lowercase letter between a and z, inclusive.” This is *only* equivalent to “any lowercase letter” if the *a* is the first and *z* is the last lowercase letter in the collating sequence.

To provide the intended capabilities in a portable way, the Subcommittee introduced a new syntactical element, namely an explicit *character class*. The definition of which characters constitute a specific character class is already present via the LC_CTYPE category of the *setlocale()* function.

The Subcommittee selected the identification of character classes by *name*, bracketed by [: and :]. A character class cannot be used as an endpoint in a range statement.

Internationalization Syntax

The Subcommittee was careful to propose changes in the regular expression syntax that minimize the impact on existing REs. In evaluating alternatives, the Subcommittee looked at ease of use (terseness, ease to remember, keyboard availability), impact on historical REs (compatibility), implementability, performance and how error-prone the syntax is likely to be (ambiguity).

The Subcommittee made the following evaluation:

- (1) Syntax changes must be limited to expressions within square brackets.
- (2) Strings or characters with special meaning must be delimited from ordinary strings, to avoid compatibility problems.
- (3) Both initial and terminating delimiter should consist of two characters, to minimize compatibility and ambiguity problems.
- (4) Outer delimiter character should be bracketing; i.e., naturally indicate initial and terminating side. Examples: { } < > ().
- (5) The brackets ([]) are, due to the special rules for “brackets within brackets,” rather unlikely to be used in the intended way (a closing bracket must precede an open bracket in the existing syntax).
- (6) To minimize ambiguity, brackets must be paired with another character. Many other symbols are already in use, either within regular expressions, or in the shell. Examples of usable characters are: = . :
- (7) Because a multicharacter collating element also can be a member of an equivalence class, different delimiters must be chosen for these two expressions. Also, the character class expression must be distinguishable from, e.g., multicharacter collating symbols; although no historical example is known to the Subcommittee, prudence dictated that character classes be given separate delimiters.
- (8) The Subcommittee selected the period as the secondary delimiter for multicharacter collating symbols.
- (9) The Subcommittee selected the equals-sign as the secondary delimiter for equivalence classes.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

- (10) The Subcommittee selected the colon as the secondary delimiter for character classes.

The specific syntax and facilities described in this clause represent a coalescence of proposals and implementations from several vendors. Due to differences in facilities and syntax, it was not possible to take one implementation and codify it. There are now several implementations closely patterned on the existing proposal.

The facilities presented in this clause are described in a manner that does not preclude their use with multibyte character sets. However, no attempt has been made to include facilities specifically intended for such character sets.

The definitions of character classes is tied to the LC_CTYPE definition. The set of character classes defined in the C Standard {7} represents the minimum set of character classes required worldwide, i.e., those required by all implementations. It is the working group's belief that local standards bodies, as well as individual vendors, will provide extensions to the standard in these areas, for instance to provide, for example, Kanji character classes.

In many historical implementations, an *invalid range* is treated as if it consisted of the endpoints only. For example, `[z-a]` is treated as `[za]`. Some implementations treat the above range as `[z]`, and others as `[-az]`. Neither is correct, and the working group decided that this should be treated as an error.

It was proposed that the syntax for bracket expressions be simplified such that the “extra” brackets are not needed if the bracket expression only consists of a character class, an equivalence class, or a collating symbol: “`[:alpha:]`” instead of “`[[:alpha:]]`”. To ensure unambiguity, if a bracket expression starts with `:`, `=`, or `.`, then it cannot contain a class expression or a collating symbol (or duplicated characters). In addition, it was also proposed that only valid class or collating symbol expressions be accepted: e.g., `[[:ctrl:]]` is an invalid expression. The working group rejected the proposal. While the syntax `[:alpha:]` may be intuitive to some, the proposal does not allow, e.g., `[:digit:].ch.]`. The alternative, to require additional brackets for the latter case would probably cause more errors than the historical syntax. Requiring erroneous class expressions or collating symbols to make the regular expression invalid may minimize the risks for inadvertent spelling errors. However, at this point it was judged that this would reduce consensus.

Consideration was given to eliminating the `[.ch.]` syntax and providing that collating element should be recognized as such both inside and outside bracket expressions. In addition, consideration was given to defining character classes such that collating elements are included. The working group rejected these proposals. The `[.ch.]` syntax is only required inside bracket expressions due to the fact that a bracket expression historically only matched a single character. If `ch` is a collating element, a range `[a-z]` (if “`ch`” falls within it) matches `ch`. Outside brackets, an expression `ch` is treated as two concatenated characters, matching the string “`ch`”. The `[.ch.]` expression is intended to allow the specification of a multicharacter collating element separately from ranges in a bracket expression. Character classes are not intended to include collating elements; there is no

requirement that all characters in a multicharacter collating element belong to the same character class (for instance “Ch” is “alpha” but neither “upper” nor “lower”). Introducing collating elements in character classes would be nonintuitive.

It was suggested that, because ranges may or may not be meaningful (or even accepted) based on the current collating sequence, they should be eliminated from the syntax (or at least marked obsolescent). It was suggested that, e.g., [z-a] should always be or never be an error, regardless of collating sequence. The working group did not wish to eliminate ranges from the syntax. While it is true that ranges may not be universally portable, they are nevertheless a useful and fundamental construct in regular expressions. The regular expression syntax has consciously been extended to provide both increased portability and extended local capabilities. Where supported, ranges must reflect the current collating sequence. The working group instead elected to include range expressions as an implementation requirement, but state that strictly conforming applications (but not, e.g., National-Body-conforming applications) shall not use range expressions. Treating erroneous ranges as invalid points out that these may not be portable across collating sequences; and is better than (silently) making them behave in a way contrary to the intents of the user.

Earlier drafts allowed the use of an equivalence class expression as the starting or ending point of a range expression, such as [[=e=]-f]. This now produces unspecified results because it is possible to define the equivalence class as a disjoint set of characters. This example could produce different results on various systems:

- An error.
- The equivalent of [[=e=]e-f] (which is the correct portable way to include equivalence class effects in a bracket expression).
- All of the collating elements from the lowest value found in the equivalence class, including any of the elements found between the disjoint values.

Consideration was given to saying that equivalence classes with disjoint elements produce unspecified results at the start or end of a range, but since the application cannot predict which equivalence classes are disjoint, this is no improvement over the more general statement chosen.

It was suggested that, while reference to nonprintable characters is partially supported by the proposed set of character classes, the specificity is not precise enough, and that additional character classes should be supported, e.g., [:tab:] or [:a:]. The working group rejected this proposal, because this feature would represent a substantial enhancement to the current regular expression syntax, and one that cannot be based on internationalization requirements. It is judged that its inclusion would reduce consensus. A future revision of regular expressions should study the capability to create temporary character classes for use in regular expressions; a “character class macro facility.”

2.8.6.3.3 BREs Matching Multiple Characters Rationale. *(This subclause is not a part of P1003.2)*

The limit of nine backreferences to subexpressions in the RE is based on the use of a single digit identifier; increasing this to multiple digits would break historical applications. This does not imply that only nine subexpressions are allowed in REs. The following is a valid BRE with ten subexpressions:

```
\(\(\(ab\)*c\)*d\)\(ef\)*\(gh\)\{2\}\(ij\)*\(kl\)*\(mn\)*\(op\)*\(qr\)*
```

The working group regards the common current behavior, which supports `\n*`, but not `\n\{min,max\}`, or `\(...\)*`, or `\(...\)\{min,max\}`, as a nonintentional result of a specific implementation, and supports both duplication and interval expressions following subexpressions and backreferences.

2.8.6.3.4 Expression Anchoring Rationale. *(This subclause is not a part of P1003.2)*

Often, the dollar-sign is viewed as matching the ending `<newline>` in text files. This is not strictly true; the `<newline>` is typically eliminated from the strings to be matched and the dollar-sign matches the terminating null character.

The ability of `^`, `$`, and `*` to be nonspecial in certain circumstances may be confusing to some programmers, but this situation was changed only in a minor way from historical practice to avoid breaking many existing scripts. Some consideration was given to making the use of the anchoring characters undefined if not escaped and not at the beginning or end of strings. This would cause a number of historical BREs, such as `2^10`, `$HOME`, and `$1.35`, which relied on the characters being treated literally, to become invalid.

However, one relatively uncommon case was changed to allow an extension used on some implementations. Historically, the BREs `^foo` and `\(^foo\)` did not match the same string, despite the general rule that subexpressions and entire BREs match the same strings. To achieve balloting consensus, POSIX.2 has allowed an extension on some systems to treat these two cases in the same way by declaring that anchoring *may* occur at the beginning or end of a subexpression. Therefore, portable BREs that require a literal circumflex at the beginning or a dollar-sign at the end of a subexpression must escape them. Note that a BRE such as `a\(^bc\)` will either match `a^bc` or nothing on different systems under the POSIX.2 rules.

ERE anchoring has been different from BRE anchoring in all historical systems. An unescaped anchor character has never matched its literal counterpart outside of a bracket expression. Some systems treated `foo$bar` as a valid expression that never matched anything, others treated it as invalid. POSIX.2 mandates the former, valid unmatched behavior.

Some systems have extended the BRE syntax to add alternation. For example, the subexpression `\(foo$|bar\)` would match either `foo` at the end of the string or `bar` anywhere. The extension is triggered by the use of the undefined `\|` sequence. Because the BRE is undefined for portable scripts, the extending system is free to make other assumptions, such as that the `$` represents the end-of-line anchor in the middle of a subexpression. If it were not for the extension, the

4637 \$ would match a literal dollar-sign under the POSIX.2 rules.

1

4638 **2.8.6.4 Extended Regular Expressions Rationale.** *(This subclause is not a part of*
 4639 *P1003.2)*

4640 As with basic regular expressions, the working group decided to make the
 4641 interpretation of escaped ordinary characters undefined.

4642 The right-parenthesis is not listed as an ERE special character because it is only
 4643 special in the context of a preceding left-parenthesis. If found without a preced-
 4644 ing left-parenthesis, the right-parenthesis has no special meaning.

1

1

1

4645 Based on objections in several ballots, the *interval expression*, $\{m,n\}$, has been
 4646 added to extended regular expressions. Historically, the interval expression has
 4647 only been supported in some extended regular expression implementations. The
 4648 working group estimated that the addition of interval expressions to extended
 4649 regular expressions would not decrease consensus, and would also make basic
 4650 regular expressions more of a subset of extended regular expressions than in
 4651 many historical implementations.

4652 It was suggested that, in addition to interval expressions, backreferences ($\backslash n$)
 4653 also should be added to extended regular expressions. This was rejected by the
 4654 working group as likely to decrease consensus.

4655 In historical implementations, multiple duplication symbols are usually inter-
 4656 preted from left to right and treated as additive. As an example, $a+*b$ matches
 4657 zero or more instances of a followed by $a b$. In POSIX.2, multiple duplication sym-
 4658 bols are undefined; i.e., they cannot be relied upon for portable applications. One
 4659 reason for this is to provide some scope for future enhancements; the current syn-
 4660 tax is very crowded.

4661 The precedence of operations differs between EREs and those in `lex`; in `lex`, for
 4662 historical reasons, interval expressions have a lower precedence than concatena-
 4663 tion.

4664 **2.8.6.5 Regular Expression Grammar Rationale.** *(This subclause is not a part of*
 4665 *P1003.2)*

4666 None.

2.9 Dependencies on Other Standards

2.9.1 Features Inherited from POSIX.1

This subclause describes some of the features provided by POSIX.1 {8} that are assumed to be globally available by all systems conforming to POSIX.2. This subclause does not attempt to detail all of the POSIX.1 {8} features that are required by all of the utilities and functions defined in this standard; the utility and function descriptions point out additional functionality required to provide the corresponding specific features needed by each.

The following subclauses describe frequently used concepts. Utility and function description statements override these defaults when appropriate.

2.9.1.0.1 Features Inherited from POSIX.1 Rationale. *(This subclause is not a part of P1003.2)*

It has been pointed out that POSIX.2 assumes that a lot of POSIX.1 {8} functionality is present, but never states exactly how much. This is an attempt to clarify the assumptions.

This subclause only covers the “utilities and functions defined by this standard.” It does not mandate that the specific POSIX.1 {8} interfaces themselves be available to all application programs. A C language program compiled on a POSIX.2 system is not guaranteed that any of the POSIX.1 {8} functions are accessible. (For example, although UNIX system-based implementations of `ls` will use `stat()` to get file status, a POSIX.2 implementation of `ls` on a “LONG_NAME_OS-based” implementation might use the `get_file_attributes()` and the `get_file_time_stamps()` system calls.) POSIX.2 only requires equivalent functionality, not equal means of access. In any event, programs requiring the POSIX.1 {8} system interface should specify that they need POSIX.1 {8} conformance and not hope to achieve it by piggybacking on POSIX.2.

2.9.1.1 Process Attributes

The following process attributes, as described in POSIX.1 {8}, are assumed to be supported for all processes in POSIX.2:

controlling terminal	real group ID
current working directory	real user ID
effective group ID	root directory
effective user ID	saved set-group-ID
file descriptors	saved set-user-ID
file mode creation mask	session membership
process ID	supplementary group IDs
process group ID	

A conforming implementation may include additional process attributes.

4705 **2.9.1.1.1 Process Attributes Rationale.** *(This subclause is not a part of P1003.2)*

4706 The supplementary group IDs requirement is minimal. If {NGROUPS_MAX} is
 4707 defined to be zero, they are not required. If {NGROUPS_MAX} is greater than zero,
 4708 the supplementary group IDs are used as described in POSIX.1 {8} in various per-
 4709 mission checking operations.

4710 The saved-set-group-ID and saved-set-user-ID requirements are also minimal. If
 4711 {_POSIX_SAVED_IDS} is defined, they are required; otherwise, they are not.

4712 A controlling terminal is needed to control access to /dev/tty.

4713 The file creation semantics of POSIX.2 require the effective group ID, effective user
 4714 ID, and the file mode creation mask.

4715 Pathname resolution and access permission checks require the current working
 4716 directory, effective group ID, effective user ID, and root directory.

4717 The `kill` utility requires the effective group ID, effective user ID, process ID, pro-
 4718 cess group ID, real group ID, real user ID, saved set-group-ID, saved set-user-ID,
 4719 and session membership attributes to perform the various signal addressing and
 4720 permission checks.

4721 The `id` utility is based on the effective group ID, effective user ID, real group ID,
 4722 real user ID, and supplementary group IDs.

4723 The following process attributes described in POSIX.1 {8} do not seem to be
 4724 required by POSIX.2: parent process ID, pending signals, process signal mask,
 4725 time left until an alarm clock signal, *tms_cstime*, *tms_cutime*, *tms_stime*, and
 4726 *tms_utime*. There are probably other attributes mentioned in POSIX.1 {8} that are
 4727 not listed here.

4728 **2.9.1.2 Concurrent Execution of Processes**

4729 The following functionality of the POSIX.1 {8} *fork()* function shall be available on
 4730 all POSIX.2 conformant systems:

- 4731 (1) Independent processes shall be capable of executing independently
 4732 without either process terminating.
- 4733 (2) A process shall be able to create a new process with all of the attributes
 4734 referenced in 2.9.1.1, determined according to the semantics of a call to
 4735 the POSIX.1 {8} *fork()* function followed by a call in the child process to
 4736 one of the POSIX.1 {8} *exec* functions.

2.9.1.2.1 Concurrent Execution of Processes Rationale. *(This subclause is not a part of P1003.2)*

The historical functionality of *fork()* is required, which permits the concurrent execution of independent processes. A system with a single thread of process execution is not an appropriate base upon which to build a POSIX.2 system. (This requirement was not explicitly stated in the 1988 POSIX.1, but is included in the current POSIX.1 {8}.)

2.9.1.3 File Access Permissions

The file access control mechanism described by *file access permissions* in 2.2.2.55 applies to all files on a conforming POSIX.2 implementation.

2.9.1.3.1 File Access Permissions Rationale. *(This subclause is not a part of P1003.2)*

The entire concept of file protections and access control is assumed to be handled as in POSIX.1 {8}.

2.9.1.4 File Read, Write, and Creation

When a file is to be read or written, the file shall be opened with an access mode corresponding to the operation to be performed. If file access permissions deny access, the requested operation shall fail.

When a file that does not exist is created, the following POSIX.1 {8} features shall apply unless the utility or function description states otherwise:

- (1) The file's user ID is set to the effective user ID of the calling process.
- (2) The file's group ID is set to the effective group ID of the calling process or the group ID of the directory in which the file is being created.
- (3) The file's permission bits are set to:

$$S_IROTH \mid S_IWOTH \mid S_IRGRP \mid S_IWGRP \mid S_IRUSR \mid S_IWUSR$$
 (see POSIX.1 {8} 5.6.1.2) except that the bits specified by the process's file mode creation mask are cleared.
- (4) The *st_atime*, *st_ctime*, and *st_mtime* fields of the file shall be updated as specified in *file times update* in 2.2.2.69.
- (5) If the file is a directory, it shall be an empty directory; otherwise the file shall have length zero.
- (6) Unless otherwise specified, the file created shall be a regular file.

When an attempt is made to create a file that already exists, the action shall depend on the file type:

- (1) For directories and FIFO special files, the attempt shall fail and the utility shall either continue with its operation or exit immediately with a nonzero status, depending on the description of the utility.

(2) For regular files:

(a) The file's user ID, group ID, and permission bits shall not be changed.

(b) The file shall be truncated to zero length.

(c) The *st_ctime* and *st_mtime* fields shall be marked for update.

(3) For other file types, the effect is implementation defined.

When a file is to be appended, the file shall be opened in a manner equivalent to using the `O_APPEND` flag, without the `O_TRUNC` flag, in the POSIX.1 {8} *open()* call.

2.9.1.4.1 File Read, Write, and Creation Rationale. *(This subclause is not a part of P1003.2)*

Even though it might be possible for a process to change the mode of a file to match a requested operation and change the mode back to its original state after the operation is completed, utilities are not allowed to do this unless the utility description states otherwise. As an example, the `ed` utility `r` command fails if the file to be read does not exist (even though it could create the file and then read it) or the file permissions do not allow read access [even though it could use the POSIX.1 {8} *chmod()* function to make the file readable before attempting to open the file].

2.9.1.5 File Removal

When a directory that is the root directory or current working directory of any process is removed, the effect is implementation defined. If file access permissions deny access, the requested operation shall fail. Otherwise, when a file is removed:

(1) Its directory entry shall be removed from the file system.

(2) The link count of the file shall be decremented.

(3) If the file is an empty directory (see 2.2.2.43):

(a) If no process has the directory open, the space occupied by the directory shall be freed and the directory shall no longer be accessible.

(b) If one or more processes have the directory open, the directory contents shall be preserved until all references to the file have been closed.

(4) If the file is a directory that is not empty, the *st_ctime* field shall be marked for update.

(5) If the file is not a directory:

(a) If the link count becomes zero:

- 4809 [1] If no process has the file open, the space occupied by the file
4810 shall be freed and the file shall no longer be accessible.
- 4811 [2] If one or more processes have the file open, the file contents
4812 shall be preserved until all references to the file have been
4813 closed.
- 4814 (b) If the link count is not reduced to zero, the *st_ctime* field shall be
4815 marked for update.
- 4816 (6) The *st_ctime* and *st_mtime* fields of the containing directory shall be
4817 marked for update.

4818 **2.9.1.5.1 File Removal Rationale.** *(This subclause is not a part of P1003.2)*

4819 This is intended to be a summary of the POSIX.1 {8} *unlink()* and *rmdir()* require-
4820 ments needed by POSIX.2.

4821 **2.9.1.6 File Time Values**

4822 All files have the three time values described by *file times update* in 2.2.2.69.

4823 **2.9.1.6.1 File Time Values Rationale.** *(This subclause is not a part of P1003.2)*

4824 All three time stamps specified by POSIX.1 {8} are needed for utilities like *find*,
4825 *ls*, *make*, *test*, and *touch* to work as expected.

4826 **2.9.1.7 File Contents**

4827 When a reference is made to the contents of a file, *pathname*, this means the
4828 equivalent of all of the data placed in the space pointed to by *buf* when performing
4829 the *read()* function calls in the following POSIX.1 {8} operations:

```
4830     while (read (fildes, buf, nbytes) > 0)
4831         ;
```

4832 If the file is indicated by a *pathname* *pathname*, the file descriptor shall be deter-
4833 mined by the equivalent of the following POSIX.1 operation:

```
4834     fildes = open (pathname, O_RDONLY);
```

4835 The value of *nbytes* in the above sequence is unspecified; if the file is of a type
4836 where the data returned by *read()* would vary with different values, the value
4837 shall be one that results in the most data being returned.

4838 If the *read()* function calls would return an error, it is unspecified whether the
4839 contents of the file are considered to include any data from offsets in the file
4840 beyond where the error would be returned.

2.9.1.7.1 File Contents Rationale. *(This subclause is not a part of P1003.2)*

This description is intended to convey the traditional behavior for all types of files. This matches the intuitive meaning for regular files, but the meaning is not always intuitive for other types of files. In particular, for FIFOs, pipes, and terminals it must be clear that the contents are not necessarily static at the time a file is opened, but they include the data returned by a sequence of reads until end-of-file is indicated. This is why the *open()* call is specified, with the *O_NONBLOCK* flag not set.

Some files, especially character special files, are sensitive to the size of a *read()* request. The contents of the file are those resulting from proper choice of this size.

2.9.1.8 Pathname Resolution

The pathname resolution algorithm described by *pathname resolution* in 2.2.2.104 shall be used by conforming POSIX.2 implementations. See also *file hierarchy* in 2.2.2.58.

2.9.1.8.1 Pathname Resolution Rationale. *(This subclause is not a part of P1003.2)*

The whole concept of hierarchical file systems and pathname resolution is assumed to be handled as in POSIX.1 {8}.

2.9.1.9 Changing the Current Working Directory

When the current working directory (see 2.2.2.159) is to be changed, unless the utility or function description states otherwise, the operation shall succeed unless a call to the POSIX.1 {8} *chdir()* function would fail when invoked with the new working directory pathname as its argument.

2.9.1.9.1 Changing the Current Working Directory Rationale. *(This subclause is not a part of P1003.2)*

This subclause covers the access permissions and pathname structures involved with changing directories, such as with *cd* or (the UPE-extended) *mailx* utilities.

2.9.1.10 Establish the Locale

The functionality of the POSIX.1 {8} *setlocale()* function is assumed to be available on all POSIX.2 conformant systems; i.e., utilities that require the capability of establishing an international operating environment shall be permitted to set the specified category of the international environment.

4873 **2.9.1.10.1 Establish the Locale Rationale.** *(This subclause is not a part of P1003.2)*

4874 The entire concept of locale categories such as the **LC_*** variables along with any
4875 implementation-defined categories is assumed to be handled as in POSIX.1 {8}.

4876 **2.9.1.11 Actions Equivalent to POSIX.1 Functions**

4877 Some utility descriptions specify that a utility performs actions equivalent to a
4878 POSIX.1 {8} function. Such specifications require only that the external effects be
4879 equivalent, not that any effect within the utility and visible only to the utility be
4880 equivalent.

4881 **2.9.1.11.1 Actions Equivalent to POSIX.1 Functions Rationale.** *(This subclause 4882 is not a part of P1003.2)*

4883 An objection was received to an earlier draft that said this approach of equivalent
4884 functions was unreasonable, as the reader (and the person writing a test suite)
4885 would be responsible for interpreting which portions of POSIX.1 {8} were included
4886 and which were not. For example, would such intermediate effects as the setting
4887 of *errno* be required if the related POSIX.1 {8} function called for that? The
4888 answer is no: this standard is only concerned with the end results of functions
4889 against the file system and the environment, and not any intermediate values or
4890 results visible only to the programmer using the POSIX.1 {8} function in a C (or
4891 other high-level language) program.

4892 **2.9.2 Concepts Derived from the C Standard**

4893 Some of the standard utilities perform complex data manipulation using their
4894 own procedure and arithmetic languages, as defined in their Extended Descrip-
4895 tion or Operands subclauses. Unless otherwise noted, the arithmetic and seman-
4896 tic concepts (precision, type conversion, control flow, etc.) are equivalent to those
4897 defined in the C Standard {7}, as described in the following subclauses. Note that
4898 there is no requirement that the standard utilities be implemented in any partic-
4899 ular programming language.

4900 **2.9.2.0.1 Concepts Derived from the C Standard Rationale.** *(This subclause is 4901 not a part of P1003.2)*

4902 This subclause was introduced to answer complaints that there was insufficient
4903 detail presented by such utilities as *awk* or *sh* about their procedural control
4904 statements and their methods of performing arithmetic functions. Earlier drafts,
4905 derived heavily from the original manual pages, contained statements such as
4906 “*for* loops similar to the C Standard {7},” which was good enough for a general
4907 understanding, but insufficient for a real implementation.

4908 The C Standard {7} was selected as a model because most historical implementa-
4909 tions of the standard utilities were written in C. Thus, it is more likely that they
4910 will act in a manner desired by POSIX.2 without modification.

Using the C Standard {7} is primarily a notational convenience, so the many “little languages” in POSIX.2 would not have to be rigorously described in every aspect. Its selection does not require that the standard utilities be written in Standard C; they could be written in common-usage C, Ada, Pascal, assembler language, or anything else.

The sizes of the various numeric values refer to C-language datatypes that are allowed to be different sizes by the C Standard {7}. Thus, like a C-language application, a shell application cannot rely on their exact size. However, it can rely on their minimum sizes expressed in the C Standard {7}, such as {LONG_MAX} for a *long* type.

2.9.2.1 Arithmetic Precision and Operations

Integer variables and constants, including the values of operands and option-arguments, used by the standard utilities shall be implemented as equivalent to the C Standard {7} *signed long* data type; floating point shall be implemented as equivalent to the C Standard {7} *double* type. Conversions between types shall be as described in the C Standard {7}. All variables shall be initialized to zero if they are not otherwise assigned by the application’s input.

Arithmetic operators and functions shall be implemented as equivalent to those in the cited C Standard {7} section, as listed in Table 2-14.

The evaluation of arithmetic expressions shall be equivalent to that described in the C Standard {7} section 3.3 Expressions.

2.9.2.2 Mathematic Functions

Any mathematic functions with the same names as those in the C Standard {7}’s sections:

4.5 *Mathematics* <math.h>

4.10.2 *Pseudo-random sequence generation functions*

shall be implemented to return the results equivalent to those returned from a call to the corresponding C function described in the C Standard {7}.

Table 2-14 – C Standard Operators and Functions

Operation	C Standard {7} Equivalent Reference
()	3.3.1 Primary Expressions
postfix ++ postfix --	3.3.2 Postfix Operators
unary + unary – prefix ++ prefix -- ~ ! sizeof ()	3.3.3 Unary Operators
* / %	3.3.5 Multiplicative Operators
+ –	3.3.6 Additive Operators
<< >>	3.3.7 Bitwise Shift Operators
<, <= >, >=	3.3.8 Relational Operators
== !=	3.3.9 Equality Operators
&	3.3.10 Bitwise AND Operator
^	3.3.11 Bitwise Exclusive OR Operator
	3.3.12 Bitwise Inclusive OR Operator
&&	3.3.13 Logical AND Operator
	3.3.14 Logical OR Operator
expr?expr:expr	3.3.15 Conditional Operator
=, *=, /=, %=, +=, -= <<=, >>=, &=, ^=, =	3.3.16 Assignment Operators
if () if () ... else switch ()	3.6.4 Selection Statements
while () do ... while () for ()	3.6.5 Iteration Statements
goto continue break return	3.6.6 Jump Statements

2.10 Utility Conventions

2.10.1 Utility Argument Syntax

This subclause describes the argument syntax of the standard utilities and introduces terminology used throughout the standard for describing the arguments processed by the utilities.

Within the standard, 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 3.9.1):

```
utility_name [-a] [-b] [-c option_argument] [-d | -e]
              [-f option_argument] [operand ... ]
```

The notation used for the Synopsis subclauses imposes requirements on the implementors of the standard utilities and provides a simple reference for the reader of the standard.

- (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 `<blanks>`, 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 2.10.2 require that the option be a separate argument from its option-argument, but there are some exceptions in this standard 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 `[-f option_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 requirements on conforming applications, a conforming implementation shall permit, but shall not require, an application to specify options and option-arguments as separate arguments whether or not a `<space>` is shown on the synopsis line.
 - (d) A standard utility may also be implemented to operate correctly when the required separation into multiple arguments is violated by a nonconforming application.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

- (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 subclause, or unless the exception in 2.10.2 guideline 11 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 shall never be included 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 shall be interpreted as a decimal integer.
- numerals in the range 0 to 2 147 483 647 shall be 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 shall be syntactically recognized as numeric values.

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 this standard, 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. The [and] symbols shall never be included in data submitted to the utility.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

- (8) Arguments separated by the | vertical bar notation are mutually exclusive. The | symbols shall never be included 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 -e [-b] [operand ...]
```

When multiple synopsis lines are given for a utility, that 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 subclause, the relationships between arguments depicted in the Synopsis subclauses 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 [] 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 ...]
```

1

```
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 subclause of the utility. (See also Guideline 11 in 2.10.2.) In the first example, each option-argument requires a preceding -f and at least one -f *option_argument* must be given.

1

1

- (10) When the synopsis line is too long to be printed on a single line in this document, the indented lines following the initial line are continuation lines. An actual use of the command would appear on a single logical line.

2.10.1.1 Utility Argument Syntax Rationale. *(This subclause is not a part of P1003.2)*

This is the subclause where the definitions of *option*, *option-argument*, and *operand* come together.

The working group felt that recent trends toward diluting the Synopsis subclauses of historical manual pages to something like:

```
command [options] [operands]
```

were a disservice to the reader. Therefore, considerable effort was placed into rigorous definitions of all the command line arguments and their interrelationships. The relationships depicted in the Synopses are normative parts of this standard; this information is sometimes repeated in textual form, but that is only for clarity within context.

The use of “undefined” for conflicting argument usage and for repeated usage of the same option is meant to prevent portable applications from using conflicting arguments or repeated options, unless specifically allowed, as is the case with `ls` (which allows simultaneous, repeated use of the `-C`, `-l`, and `-1` options). Many historical implementations will tolerate this usage, choosing either the first or the last applicable argument, and this tolerance can continue, but portable applications cannot rely upon it. (Other implementations may choose to print usage messages instead.)

The use of “undefined” for conflicting argument usage also allows an implementation to make reasonable extensions to utilities where the implementor considers mutually exclusive options according to POSIX.2 to have a sensible meaning and result.

POSIX.2 does not define the result of a utility when an option-argument or operand is not followed by ellipses and the application specifies more than one of that option-argument or operand. This allows an implementation to define valid (although nonstandard) behavior for the utility when more than one such option or operand are specified.

Allowing `<blank>s` after an option (i.e., placing an option and its option-argument into separate argument strings) when the standard does not require it encourages portability of users, while still preserving backward compatibility of scripts. Inserting `<blank>s` between the option and the option-argument is preferred; however, historical usage has not been consistent in this area; therefore, `<blank>s` are required to be handled by all implementations, but implementations are also allowed to handle the historical syntax. Another justification for selecting the multiple-argument method was that the single-argument case is inherently ambiguous when the option-argument can legitimately be a null string.

Wording was also added to explicitly state that digits are permitted as operands and option-arguments. The lower and upper bounds for the values of the numbers used for operands and option-arguments were derived from the C Standard {7} values for {LONG_MIN} and {LONG_MAX}. The requirement on the standard utilities is that numbers in the specified range do not cause a syntax

5141 error although the specification of a number need not be semantically correct for a
5142 particular operand or option-argument of a utility. For example, the specification
5143 of `dd obs=3000000000` would yield undefined behavior for the application and
5144 would be a syntax error because the number 3 000 000 000 is outside of the range
5145 $-2\,147\,483\,647$ to $+2\,147\,483\,647$. On the other hand, `dd obs=2000000000` may
5146 cause some error, such as “blocksize too large,” rather than a syntax error.

2.10.2 Utility Syntax Guidelines

The following guidelines are established for the naming of utilities and for the specification of options, option-arguments, and operands. Clause 7.5 describes a function that assists utilities in handling options and operands that conform to these guidelines.

Operands and option-arguments can contain characters not specified in 2.4.

The guidelines are intended to provide guidance to the authors of future utilities. Some of the standard utilities do not conform to all of these guidelines; in those cases, the Options subclauses 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) from the set described in 2.4 and digits only.

Guideline 3: Each option name should be a single alphanumeric character (the alnum character classification) from the set described in 2.4. The `-W` (capital-W) option shall be reserved for vendor extensions.

NOTE: The other alphanumeric characters are subject to standardization in the future, based on historical usage. Implementors should be aware that future POSIX working groups may offer little sympathy to vendors with isolated extensions in conflict with future drafts.

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 2.10.1, item (2).

Guideline 7: Option-arguments should not be optional.

Guideline 8: When multiple option-arguments are specified to follow a single option, they should be presented as a single argument, using commas within that argument or `<blank>`s within that argument to separate them.

Guideline 9: All options should precede operands on the command line.

Guideline 10: The argument `"--"` should be accepted as a delimiter indicating the end of options. Any following arguments should be treated as operands, even if they begin with the `'-'` character. The `"--"` argument should not be used as an option or as an operand.

Guideline 11: The order of different options relative to one another should not matter, unless the options are documented as mutually

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

exclusive and such an option is documented to override any incompatible options preceding it. If an option that has option-arguments is repeated, the option and option-argument combinations should be interpreted in the order specified on the command line.

Guideline 12: The order of operands may matter and position-related interpretations should be determined on a utility-specific basis.

Guideline 13: For utilities that use operands to represent files to be opened for either reading or writing, the "-" operand should be used only to mean standard input (or standard output when it is clear from context that an output file is being specified).

Any utility claiming conformance to these guidelines shall conform completely to these guidelines, as if these guidelines contained the term "shall" instead of "should," except that the utility is permitted to accept usage in violation of these guidelines for backward compatibility as long as the required form is also accepted.

Guidelines 1 and 2 are offered as guidance for locales using Latin alphabets. No recommendations are made by this standard concerning utility naming in other locales.

2.10.2.1 Utility Syntax Guidelines Rationale. *(This subclause is not a part of P1003.2)*

This subclause is based on the rules listed in the *SVID*. It was included for two reasons:

- (1) The individual utility descriptions in Sections 4, 5, and 6, and Annexes A and C needed a set of common (although not universal) actions on which they could anchor their descriptions of option and operand syntax. Most of the standard utilities actually do use these guidelines, and many of their historical implementations use the *getopt()* function for their parsing. Therefore, it was simpler to cite the rules and merely identify exceptions.
- (2) Writers of portable applications need suggested guidelines if the POSIX community is to avoid the chaos of historical UNIX system command syntax.

It is recommended that all *future* utilities and applications use these guidelines to enhance "user portability." The fact that some historical utilities could not be changed (to avoid breaking existing applications) should not deter this future goal.

The voluntary nature of the guidelines is highlighted by repeated uses of the word *should* throughout. This usage should not be misinterpreted to imply that utilities that claim conformance in their Options subclauses do not always conform.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

Guideline 2 recommends the naming of utilities. In 3.9.1, it is further stated that a command used in the shell command language cannot be named with a trailing colon.

Guideline 3 was changed to allow alphanumeric characters (letters and digits) from the character set to allow compatibility with historical usage. Historical practice allows the use of digits wherever practical; and there are no portability issues that would prohibit the use of digits. In fact, from an internationalization viewpoint, digits (being nonlanguage dependent) are preferable over letters (a “-2” is intuitively self-explanatory to any user, while in the “-f *filename*” the letter f is a mnemonic aid only to speakers of Latin based languages where “filename” happens to translate to a word that begins with f. Since guideline 3 still retains the word “single,” multidigit options are not allowed. Instances of historical utilities that used them have been marked obsolescent in this standard, with the numbers being changed from option names to option-arguments.

It is difficult to come up with a satisfactory solution to the problem of namespace in option characters. When the POSIX.2 group desired to extend the historical `cc` utility to accept C Standard [7] programs, it found that all of the portable alphabet was already in use by various vendors. Thus, it had to devise a new name, `c89`, rather than something like `cc -X`. There were suggestions that implementors be restricted to providing extensions through various means (such as using a plus-sign as the option delimiter or using option characters outside the alphanumeric set) that would reserve all of the remaining alphanumeric characters for future POSIX standards. These approaches were resisted because they lacked the historical style of UNIX. Furthermore, if a vendor-provided option should become commonly used in the industry, it would be a candidate for standardization. It would be desirable to standardize such a feature using existing practice for the syntax (the semantics can be standardized with any syntax). This would not be possible if the syntax was one reserved for the vendor. However, since the standardization process may lead to minor changes in the semantics, it may prove to be better for a vendor to use a syntax that will not be affected by standardization. As a compromise, the following statements are made by the developers of POSIX.2:

- In future revisions to this standard, and in other POSIX standards, every attempt will be made to develop new utilities and features that conform to the Utility Syntax Guidelines.
- Future extensions and additions to POSIX standards will not use the -W (capital W) option. This option is forever reserved to implementors for extensions, in a manner reminiscent of the option’s use in historical versions of the `cc` utility. The other alphanumeric characters are subject to standardization in the future, based on historical usage.

Implementors should be cognizant of these intentions and aware that future POSIX working groups will offer little sympathy to vendors with extensions in conflict with future drafts. In the first version of POSIX.2, vendors held a virtual veto power when conflicts arose with their extensions; in the future, POSIX working groups may be less concerned about preserving isolated extensions that conflict with these statements of intent.

Guideline 8 includes the concept of comma-separated lists in a single argument. It is up to the utility to parse such a list itself because *getopt()* just returns the single string. This situation was retained so that certain historical utilities wouldn't violate the guidelines. Applications preparing for international use should be aware of an occasional problem with comma-separated lists: in some locales, the comma is used as the radix character. Thus, if an application is preparing operands for a utility that expects a comma-separated lists, it should avoid generating noninteger values through one of the means that is influenced by setting the **LC_NUMERIC** variable [such as *awk*, *bc*, *printf*, or *printf()*].

Applications calling any utility with a first operand starting with "-" should usually specify "--", as indicated by Guideline 10, to mark the end of the options. This is true even if the Synopsis in this standard does not specify any options; implementations may provide options as extensions to this standard. The standard utilities that do not support Guideline 10 indicate that fact in the Options subclause of the utility description.

Guideline 11 was modified to clarify that the order of different options should not matter relative to one another. However, the order of repeated options that also have option-arguments may be significant; therefore, such options are required to be interpreted in the order that they are specified. The *make* utility is an instance of a historical utility that uses repeated options in which the order is significant. Multiple files are specified by giving multiple instances of the *-f* option, for example:

```
make -f common_header -f specific_rules target
```

Guideline 13 does not imply that all of the standard utilities automatically accept the operand "-" to mean standard input or output, nor does it specify the actions of the utility upon encountering multiple "-" operands. It simply says that, by default, "-" operands shall not be used for other purposes in the file reading/writing [but not *stat()*ing, *unlink()*ing, *touching*, etc.] utilities. All information concerning actual treatment of the "-" operand is found in the individual utility clauses.

An area of concern that was expressed during the balloting process was that as implementations mature implementation-defined utilities and implementation-defined utility options will result. The notion was expressed that there needed to be a standard way, say an environment variable or some such mechanism, to identify implementation-defined utilities separately from standard utilities that may have the same name. It was decided that there already exist several ways of dealing with this situation and that it is outside of the scope of the standard to attempt to standardize in the area of nonstandard items. A method that exists on some historical implementations is the use of the so-called */local/bin* or */usr/local/bin* directory to separate local or additional copies or versions of utilities. Another method that is also used is to isolate utilities into completely separate domains. Still another method to ensure that the desired utility is being used is to request the utility by its full pathname. There are, to be sure, many approaches to this situation; the examples given above serve to illustrate that there is more than one.

2.11 Utility Description Defaults

This clause describes all of the subclauses used within the utility clauses in Section 4 and the other sections that describe standard utilities. It describes:

- (1) Intended usage of the subclause.
- (2) Global defaults that affect all the standard utilities.

2.11.0.1 Utility Description Defaults Rationale. *(This subclause is not a part of P1003.2)*

This clause is arranged with headings in the same order as all the utility descriptions. It is a collection of related and unrelated information concerning:

- (1) The default actions of utilities.
- (2) The meanings of notations used in the standard that are specific to individual utility subclauses.

Although this material may seem out of place in Section 2, it is important that this information appear before any of the utilities to be described later. Unfortunately, since the utilities are split into multiple major sections (chapters), this information could not be placed into any one of those sections without confusing cross references.

2.11.1 Synopsis

The Synopsis subclause summarizes the syntax of the calling sequence for the utility, including options, option-arguments, and operands. Standards for utility naming are described in 2.10.2; for describing the utility's arguments in 2.10.1.

2.11.2 Description

The Description subclause describes the actions of the utility. If the utility has a very complex set of subcommands or its own procedural language, an Extended Description subclause is also provided. Most explanations of optional functionality are omitted here, as they are usually explained in the Options subclause.

Some utilities in this standard are described in terms of equivalent POSIX.1 {8} functionality. As explained in 1.1, a fully conforming POSIX.1 {8} base is not a prerequisite for this standard. When specific functions are cited, the underlying operating system shall provide equivalent functionality and all side effects associated with successful execution of the function. The treatment of errors and intermediate results from the individual functions cited are generally not specified by this standard. See the utility's Exit Status and Consequences of Errors subclauses for all actions associated with errors encountered by the utility.

2.11.3 Options

The Options subclause describes the utility options and option-arguments, and how they modify the actions of the utility. Standard utilities that have options either fully comply with the 2.10.2 or describe all deviations. Apparent disagreements between functionality descriptions in the Options and Description (or Extended Description) subclauses are always resolved in favor of the Options subclause.

Each Options subclause that uses the phrase “The ... utility shall conform to the utility argument syntax guidelines ...” refers only to the use of the utility as specified by this standard; implementation extensions should also conform to the guidelines, but may allow exceptions for historical practice.

Unless otherwise stated in the utility description, when given an option unrecognized by the implementation, or when a required option-argument is not provided, standard utilities shall issue a diagnostic message to standard error and exit with a nonzero exit status.

Default Behavior: When this subclause is listed as “None,” it means that the implementation need not support any options. Standard utilities that do not accept options, but that do accept operands, shall recognize “--” as a first argument to be discarded.

2.11.3.1 Options Rationale. *(This subclause is not a part of P1003.2)*

Although it has not always been possible, the working group has tried to avoid repeating information and therefore reduced the risk that the duplicate explanations are somehow modified to be out of sync.

The requirement for recognizing -- is because portable applications need a way to shield their operands from any arbitrary options that the implementation may provide as an extension. For example, if the standard utility `foo` is listed as taking no options, and the application needed to give it a pathname with a leading hyphen, it could safely do it as:

```
foo -- -myfile
```

and avoid any problems with `-m` used as an extension.

2.11.4 Operands

The Operands subclause describes the utility operands, and how they affect the actions of the utility. Apparent disagreements between functionality descriptions in the Operands and Description (or Extended Description) subclauses are always resolved in favor of the Operands subclause.

If an operand naming a file can be specified as `-`, which means to use the standard input instead of a named file, this shall be explicitly stated in this subclause. Unless otherwise stated, the use of multiple instances of `-` to mean standard input in a single command produces unspecified results.

Unless otherwise stated, the standard utilities that accept operands shall process those operands in the order specified in the command line.

Default Behavior: When this subclause is listed as “None,” it means that the implementation need not support any operands.

2.11.4.1 Operands Rationale. *(This subclause is not a part of P1003.2)*

This usage of `-` is never shown in the Synopsis. Similarly, this usage of `--` is never shown.

The requirement for processing operands in command line order is to avoid a “WeirdNIX” utility that might choose to sort the input files alphabetically, by size, or by directory order. Although this might be acceptable for some utilities, in general the programmer has a right to know exactly what order will be chosen.

Some of the standard utilities take multiple *file* operands and act as if they were processing the concatenation of those files. For example,

```
asa file1 file2          and          cat file1 file2 | asa
```

have similar results when questions of file access, errors, and performance are ignored. Other utilities, such as `grep` or `wc`, have completely different results in these two cases. This latter type of utility is always identified in its Description or Operands subclauses, whereas the former is not. Although it might be possible to create a general assertion about the former case, the following points must be addressed:

- Access times for the files might be different in the operand case versus the `cat` case.
- The utility may have error messages that are cognizant of the input file name and this added value should not be suppressed. (As an example, `awk` sets a variable with the file name at each file boundary.)

2.11.5 External Influences

The External Influences subclause describes all input data that is specified by the invoker, data received from the environment, and other files or databases that may be used by the utility. There are four subclauses that contain all the substantive information about external influences; because of this, this level of header is always left blank.

Certain of the standard utilities describe how they can invoke other utilities or applications, such as by passing a command string to the command interpreter. The external requirements of such invoked utilities are not described in the subclause concerning the standard utility that invokes them.

2.11.5.1 Standard Input

The Standard Input subclause describes the standard input of the utility. This subclause is frequently merely a reference to the following subclause, because many utilities treat standard input and input files in the same manner. Unless otherwise stated, all restrictions described in Input Files apply to this subclause as well.

Use of a terminal for standard input may cause any of the standard utilities that read standard input to stop when used in the background. For this reason, applications should not use interactive features in scripts to be placed in the background.

The specified standard input format of the standard utilities shall not depend on the existence or value of the environment variables defined in this standard, except as provided by this standard.

Default Behavior: When this subclause is listed as “None,” it means that the standard input shall not be read when the utility is used as described by this standard.

2.11.5.1.1 Standard Input Rationale. *(This subclause is not a part of P1003.2)*

This subclause was globally renamed from Standard Input Format in previous drafts to better reflect its role in describing the existence and usage of the file, in addition to its format.

2.11.5.2 Input Files

The Input Files subclause describes the files, other than the standard input, used as input by the utility. It includes files named as operands and option-arguments as well as other files that are referred to, such as startup/initialization files, databases, etc. Commonly-used files are generally described in one place and cross-referenced by other utilities.

Some of the standard utilities, such as filters, process input files a line or a block at a time and have no restrictions on the maximum input file size. Some utilities may have size limitations that are not as obvious as file space or memory limitations. Such limitations should reflect resource limitations of some sort, not arbitrary limits set by implementors. Implementations shall define in the conformance documentation those utilities that are limited by constraints other than file system space, available memory, and other limits specifically cited by this standard, and identify what the constraint is, and indicate a way of estimating when the constraint would be reached. Similarly, some utilities descend the directory tree (recursively). Implementations shall also document any limits that they may have in descending the directory tree that are beyond limits cited by this standard.

When a standard utility reads a seekable input file and terminates without an error before it reaches end-of-file, the utility shall ensure that the file offset in the open file description is properly positioned just past the last byte processed by the

5470 utility. For files that are not seekable, the state of the file offset in the open file 1
 5471 description for that file is unspecified. 1

5472 When an input file is described as a *text file*, the utility produces undefined
 5473 results if given input that is not from a text file, unless otherwise stated. Some
 5474 utilities (e.g., `make`, `read`, `sh`, etc.) allow for continued input lines using an
 5475 escaped `<newline>` convention; unless otherwise stated, the utility need not be
 5476 able to accumulate more than `{LINE_MAX}` bytes from a set of multiple, continued
 5477 input lines. If a utility using the escaped `<newline>` convention detects an end-
 5478 of-file condition immediately after an escaped `<newline>`, the results are
 5479 unspecified.

5480 Record formats are described in a notation similar to that used by the C language
 5481 function, `printf()`. See 2.12 for a description of this notation.

5482 **Default Behavior:** When this subclause is listed as “None,” it means that no
 5483 input files are required to be supplied when the utility is used as described by this
 5484 standard.

5485 **2.11.5.2.1 Input Files Rationale.** (*This subclause is not a part of P1003.2*)

5486 This subclause was globally renamed from Input File Formats in previous drafts
 5487 to better reflect its role in describing the existence and usage of the files, in addi-
 5488 tion to their format.

5489 The description of file offsets answers the question: Are the following three com- 1
 5490 mands equivalent? 1

```
5491     tail -n +2 file 1
5492     (sed -n 1q; cat) < file 1
5493     cat file | (sed -n 1q; cat) 1
```

5494 The answer is that a conforming application cannot assume they are equivalent. 1
 5495 The second command is equivalent to the first only when the file is seekable. In 1
 5496 the third command, if the file offset in the open file description were not 1
 5497 unspecified, `sed` would have to be implemented so that it read from the pipe one 1
 5498 byte at a time or it would have to employ some method to seek backwards on the 1
 5499 pipe. Such functionality is not defined currently in POSIX.1 {8} and does not exist 1
 5500 on all historical systems. Other utilities, such as `head`, `read`, and `sh`, have simi- 1
 5501 lar properties, so the restriction is described globally in this clause. A future revi- 1
 5502 sion to this standard may require that the standard utilities leave the file offset in 1
 5503 a consistent state for pipes as well as regular files. 1

5504 The description of conformance documentation about file sizes follows many
 5505 changes of direction by the working group. Originally, there appeared a limit,
 5506 `{ED_FILE_MAX}`, that hoped to impose a minimum file size on `ed`, which has been
 5507 historically limited to relatively small files. This received objections from various
 5508 members who said that such a limit merely invited sloppy programming; there
 5509 should be no limits to a “well-written” `ed`. Thus, Draft 8 removed the limit and
 5510 inserted rationale that this meant `ed` would have to process files of virtually
 5511 unlimited size. (Surprisingly, no objections or comments were received about that
 5512 sentence.) However, in discussing the matter with representatives of POSIX.3, it

turned out that omitting the limit meant that a corresponding test assertion would also be omitted and no test suite could legitimately stress ed with large files. It quickly became clear that restrictions applied to other utilities as well and a solution was needed.

It is not possible for this standard to judge which utilities are in the category with arbitrary file size limits; this would impose too much on implementors. Therefore, the burden is placed on implementors to publicly document any limitations and the resulting pressure in the marketplace should keep most implementations adequate for most portable applications. Typically, larger systems would have larger limits than smaller systems, but since price typically follows function, the user can select a machine that handles his/her problems reasonably given such information. The working group considered adding a limit in 2.13.1 for every file-oriented utility, but felt these limits would not actually be used by real applications and would reduce consensus. This is particularly true for utilities, such as possibly `awk` or `yacc`, that might have rather complex limits not directly related to the actual file size.

The definition of *text file* (see 2.2.2.151) is strictly enforced for input to the standard utilities; very few of them list exceptions to the undefined results called for here. (Of course, “undefined” here does not mean that existing implementations necessarily have to change to start indicating error conditions. Conforming applications cannot rely on implementations succeeding or failing when nontext files are used.)

The utilities that allow line continuation are generally those that accept input languages, rather than pure data. It would be unusual for an input line of this type to exceed `{LINE_MAX}` bytes and unreasonable to require that the implementation allow unlimited accumulation of multiple lines, each of which could reach `{LINE_MAX}`. Thus, for a portable application the total of all the continued lines in a set cannot exceed `{LINE_MAX}`.

The format description is intended to be sufficiently rigorous to allow other applications to generate these input files. However, since `<blank>`s can legitimately be included in some of the fields described by the standard utilities, particularly in locales other than the POSIX Locale, this intent is not always realized.

2.11.5.3 Environment Variables

The Environment Variables subclause lists what variables affect the utility’s execution.

The entire manner in which environment variables described in this standard affect the behavior of each utility is described in the Environment Variables subclause for that utility, in conjunction with the global effects of the **LANG** and **LC_ALL** environment variables described in 2.6. The existence or value of environment variables described in this standard shall not otherwise affect the specified behavior of the standard utilities. Any effects of the existence or value of environment variables not described by this standard upon the standard utilities are unspecified.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

For those standard utilities that use environment variables as a means for selecting a utility to execute (such as **CC** in **make**), the string provided to the utility shall be subjected to the path search described for **PATH** in 2.6.

Default Behavior: When this subclause is listed as “None,” it means that the behavior of the utility is not directly affected by environment variables described by this standard when the utility is used as described by this standard.

2.11.5.3.1 Environment Variables Rationale. *(This subclause is not a part of P1003.2)*

The global default text about the **PATH** search is overkill in this version of POSIX.2 (prior to the UPE) because only one of the standard utilities specifies variables in this way—**make**’s $\$(CC)$, $\$(LEX)$, etc. It is described here mostly in anticipation of its heavier usage in POSIX.2a. The description of **PATH** indicates separately that names including slashes do not apply, so they do not apply here either.

2.11.5.4 Asynchronous Events

The Asynchronous Events subclause lists how the utility reacts to such events as signals and what signals are caught.

Default Behavior: When this subclause is listed as “Default,” or it refers to “the standard action for all other signals; see 2.11.5.4,” it means that the action taken as a result of the signal shall be one of the following:

- (1) The action is that inherited from the parent according to the rules of inheritance of signal actions defined in POSIX.1 {8} (see 2.9.1), or
- (2) When no action has been taken to change the default, the default action is that specified by POSIX.1 {8}, or
- (3) The result of the utility’s execution is as if default actions had been taken.

2.11.5.4.1 Asynchronous Events Rationale. *(This subclause is not a part of P1003.2)*

Because there is no language prohibiting it, a utility is permitted to catch a signal, perform some additional processing (such as deleting temporary files), restore the default signal action (or action inherited from the parent process) and resignal itself.

2.11.6 External Effects

The External Effects subclause describes the effects of the utility on the operational environment, including the file system. There are three subclauses that contain all the substantive information about external effects; because of this, this level of header is usually left blank.

Certain of the standard utilities describe how they can invoke other utilities or applications, such as by passing a command string to the command interpreter. The external effects of such invoked utilities are not described in the subclause concerning the standard utility that invokes them.

2.11.6.1 Standard Output

The Standard Output subclause describes the standard output of the utility. This subclause is frequently merely a reference to the following subclause, Output Files, because many utilities treat standard output and output files in the same manner.

Use of a terminal for standard output may cause any of the standard utilities that write standard output to stop when used in the background. For this reason, applications should not use interactive features in scripts to be placed in the background.

Record formats are described in a notation similar to that used by the C language function, *printf()*. See 2.12 for a description of this notation.

The specified standard output of the standard utilities shall not depend on the existence or value of the environment variables defined in this standard, except as provided by this standard.

Default Behavior: When this subclause is listed as “None,” it means that the standard output shall not be written when the utility is used as described by this standard.

2.11.6.1.1 Standard Output Rationale. *(This subclause is not a part of P1003.2)*

This subclause was globally renamed from Standard Output Format in previous drafts to better reflect its role in describing the existence and usage of the file, in addition to its format.

The format description is intended to be sufficiently rigorous to allow post-processing of output by other programs, particularly by an *awk* or *lex* parser.

2.11.6.2 Standard Error

The Standard Error subclause describes the standard error output of the utility. Only those messages that are purposely sent by the utility are described.

Use of a terminal for standard error may cause any of the standard utilities that write standard error output to stop when used in the background. For this reason, applications should not use interactive features in scripts to be placed in the background.

The format of diagnostic messages for most utilities is unspecified, but the language and cultural conventions of diagnostic and informative messages whose format is unspecified by this standard should be affected by the setting of **LC_MESSAGES**.

The specified standard error output of standard utilities shall not depend on the existence or value of the environment variables defined in this standard, except as provided by this standard.

Default Behavior: When this subclause is listed as “Used only for diagnostic messages,” it means that, unless otherwise stated, the diagnostic messages shall be sent to the standard error only when the exit status is nonzero and the utility is used as described by this standard.

When this subclause is listed as “None,” it means that the standard error shall not be used when the utility is used as described in this standard.

2.11.6.2.1 Standard Error Rationale. *(This subclause is not a part of P1003.2)*

This subclause was globally renamed from Standard Error Format in previous drafts to better reflect its role in describing the existence and usage of the file, in addition to its format.

This subclause does not describe error messages that refer to incorrect operation of the utility. Consider a utility that processes program source code as its input. This subclause is used to describe messages produced by a correctly operating utility that encounters an error in the program source code on which it is processing. However, a message indicating that the utility had insufficient memory in which to operate would not be described.

Some compilers have traditionally produced warning messages without returning a nonzero exit status; these are specifically noted in their subclauses. Other utilities are expected to remain absolutely quiet on the standard error if they want to return zero, unless the implementation provides some sort of extension to increase the verbosity or debugging level.

The format descriptions are intended to be sufficiently rigorous to allow post-processing of output by other programs.

2.11.6.3 Output Files

The Output Files subclause describes the files created or modified by the utility. Temporary or system files that are created for internal usage by this utility or other parts of the implementation (spool, log, audit files, etc.) are not described in this, or any, subclause. The utilities creating such files and the names of such files are unspecified. If applications are written to use temporary or intermediate files, they should use the **TMPDIR** environment variable, if it is set and represents an accessible directory, to select the location of temporary files.

Implementations shall ensure that temporary files, when used by the standard utilities, are named so that different utilities or multiple instances of the same utility can operate simultaneously without regard to their working directories, or any other process characteristic other than process ID. There are two exceptions to this requirement:

(1) Resources for temporary files other than the namespace (for example, disk space, available directory entries, or number of processes allowed) are not guaranteed.

(2) Certain standard utilities generate output files that are intended as input for other utilities, (for example, `lex` generates `lex.yy.c`) and these cannot have unique names. These cases are explicitly identified in the descriptions of the respective utilities.

Any temporary files created by the implementation shall be removed by the implementation upon a utility's successful exit, exit because of errors, or before termination by any of the `SIGHUP`, `SIGINT`, or `SIGTERM` signals, unless specified otherwise by the utility description.

Record formats are described in a notation similar to that used by the C language function, `printf()`. See 2.12 for a description of this notation.

Default Behavior: When this subclause is listed as “None,” it means that no files are created or modified as a consequence of direct action on the part of the utility when the utility is used as described by this standard. However, the utility may create or modify system files, such as log files, that are outside of the utility's normal execution environment.

2.11.6.3.1 Output Files Rationale. *(This subclause is not a part of P1003.2)*

This subclause was globally renamed from Output File Formats in previous drafts to better reflect its role in describing the existence and usage of the files, in addition to their format.

The format description is intended to be sufficiently rigorous to allow post-processing of output by other programs, particularly by an `awk` or `lex` parser.

Receipt of the `SIGQUIT` signal should generally cause termination (unless in some debugging mode) that would bypass any attempted recovery actions.

2.11.7 Extended Description

The Extended Description subclause provides a place for describing the actions of very complicated utilities, such as text editors or language processors, which typically have elaborate command languages.

Default Behavior: When this subclause is listed as “None,” no further description is necessary.

2.11.8 Exit Status

The Exit Status subclause describes the values the utility shall return to the calling program, or shell, and the conditions that cause these values to be returned. Usually, utilities return zero for successful completion and values greater than zero for various error conditions. If specific numeric values are listed in this subclause, conforming implementations shall use those values for the errors

described. In some cases, status values are listed more loosely, such as “>0.” A Strictly Conforming POSIX.2 Application shall not rely on any specific value in the range shown and shall be prepared to receive any value in the range.

Unspecified error conditions may be represented by specific values not listed in the standard.

2.11.8.1 Exit Status Rationale. *(This subclause is not a part of P1003.2)*

Note the additional discussion of exit status values in 3.8.2. It describes requirements for returning exit values > 125.

A utility may list zero as a successful return, 1 as a failure for a specific reason, and >1 as “an error occurred.” In this case, unspecified conditions may cause a 2 or 3, or other value, to be returned. A Strictly Conforming POSIX.2 Application should be written so that it tests for successful exit status values (zero in this case), rather than relying upon the single specific error value listed in the standard. In that way, it will have maximum portability, even on implementations with extensions.

The working group is aware that the general nonenumeration of errors makes it difficult to write test suites that test the *incorrect* operation of utilities. There are some historical implementations that have expended effort to provide detailed status messages and a helpful environment to bypass or explain errors, such as prompting, retrying, or ignoring unimportant syntax errors; other implementations have not. Since there is no realistic way to mandate system behavior in cases of undefined application actions or system problems—in a manner acceptable to all cultures and environments—attention has been limited to the correct operation of utilities by the conforming application. Furthermore, the portable application does not need detailed information concerning errors that it caused through incorrect usage or that it cannot correct anyway. The high degree of competition in the emerging POSIX marketplace should ensure that users requiring friendly, resilient environments will be able to purchase such without detailed specification in this standard.

There is no description of defaults for this subclause because all of the standard utilities specify something (or explicitly state “Unspecified”) for Exit Status.

2.11.9 Consequences of Errors

The Consequences of Errors subclause describes the effects on the environment, file systems, process state, etc., when error conditions occur. It does not describe error messages produced or exit status values used.

The many reasons for failure of a utility are generally not specified by the utility descriptions. Utilities may terminate prematurely if they encounter: invalid usage of options, arguments, or environment variables; invalid usage of the complex syntaxes expressed in Extended Description subclauses; difficulties accessing, creating, reading, or writing files; or, difficulties associated with the privileges of the process.

The following shall apply to each utility, unless otherwise stated:

- If the requested action cannot be performed on an operand representing a file, directory, user, process, etc., the utility shall issue a diagnostic message to standard error and continue processing the next operand in sequence, but the final exit status shall be returned as nonzero.
- If the requested action characterized by an option or option-argument cannot be performed, the utility shall issue a diagnostic message to standard error and the exit status returned shall be nonzero.
- When an unrecoverable error condition is encountered, the utility shall exit with a nonzero exit status.
- A diagnostic message shall be written to standard error whenever an error condition occurs.

Default Behavior: When this subclause is listed as “Default,” it means that any changes to the environment are unspecified.

2.11.9.1 Consequences of Errors Rationale. *(This subclause is not a part of P1003.2)*

When a utility encounters an error condition several actions are possible, depending on the severity of the error and the state of the utility. Included in the possible actions of various utilities are: deletion of temporary or intermediate work files; deletion of incomplete files; validity checking of the file system or directory.

In Draft 9, most of the Consequences of Errors subclauses were changed to “Default.” This is due to the more elaborate description of the default case now carried in this subclause and the fact that most of the standard utilities actually use that default.

2.11.10 Rationale

This subclause provides historical perspective and justification of working group actions concerning the utility.

Examples, Usage

This subclause provides examples and usage of the utility. In some cases certain characters are interpreted as special characters to the shell. In the rest of the standard, these characters are shown without escape characters or quoting (see 3.2). In all examples, however, quoting has been used, showing how sample commands (utility names combined with arguments) could be passed correctly to a shell (see `sh` in 4.56) or as a string to the `system()` function.

History of Decisions Made

This subclause provides historical perspective for decisions that were made.

Unresolved Objections

These subclauses were removed from Draft 10. The Unresolved Objections are maintained in a separate list and do not meet ISO editing requirements for an informative annex.

2.11.10.1 Rationale Rationale. *(This subclause is not a part of P1003.2)*

The Rationale subclauses will be moved to Annex E in the final POSIX.2. Some of the subheadings may be collapsed in that document; in these drafts the working group has not always been very rigorous about what is a description of usage versus a history of decisions made, for example. The final rationale will de-emphasize the chronological aspects of working group decisions.

2.12 File Format Notation

The Standard Input, Standard Output, Standard Error, Input Files, and Output Files subclauses of the utility descriptions, when provided, use a syntax to describe the data organization within the files, when that organization is not otherwise obvious. The syntax is similar to that used by the C language *printf()* function, as described in this clause. When used in Standard Input or Input Files subclauses of the utility descriptions, this syntax describes the format that could have been used to write the text to be read, not a format that could be used by the C language *scanf()* function to read the input file.

The description of an individual record is as follows:

"<format>", [<arg1>, <arg2>, ..., <argn>]

The *format* is a character string that contains three types of objects defined below:

characters Characters that are not *escape sequences* or *conversion specifications*, as described below, shall be copied to the output.

escape sequences

Represent nongraphic characters.

conversion specifications

Specifies the output format of each argument. (See below.)

The following characters have the following special meaning in the format string:

" " (An empty character position.) One or more <blank> characters.

Δ Exactly one <space> character.

The escape-sequences in Table 2-15 depict the associated action on display devices capable of the action.

Each conversion specification shall be introduced by the percent-sign character (%). After the character %, the following shall appear in sequence:

Table 2-15 – Escape Sequences

Escape Sequence	Represents Character	Terminal Action
\\	backslash	None.
\a	<alert>	Attempts to alert the user through audible or visible notification.
\b	<backspace>	Moves the printing position to one column before the current position, unless the current position is the start of a line.
\f	<form-feed>	Moves the printing position to the initial printing position of the next logical page.
\n	<newline>	Moves the printing position to the start of the next line.
\r	<carriage-return>	Moves the printing position to the start of the current line.
\t	<tab>	Moves the printing position to the next tab position on the current line. If there are no more tab positions left on the line, the behavior is undefined.
\v	<vertical tab>	Moves 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.

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 conversions (the field shall be padded with leading zeros), the number of digits to appear after the radix character for the e and f conversions, the maximum number of significant digits for the g conversion; or the maximum number of bytes to be written from a string in s conversion. The precision shall take the form of a period (.) followed by a decimal digit string; a null digit string shall be treated as zero.

conversion characters

A conversion 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 always shall begin with a sign (+ or –).

5860	<space>	If the first character of a signed conversion is not a sign, a
5861		<space> shall be prefixed to the result. This means that if the
5862		<space> and + flags both appear, the <space> flag shall be
5863		ignored.
5864	#	The value is to be converted to an “alternate form.” For c, d, i, u,
5865		and s conversions, the behavior is undefined. For o conversion, it
5866		shall increase the precision to force the first digit of the result to
5867		be a zero. For x or X conversion, a nonzero result shall have 0x or
5868		0X prefixed to it, respectively. For e, E, f, g and G conversions,
5869		the result shall always contain a radix character, even if no digits
5870		follow the radix character. For g and G conversions, trailing
5871		zeroes shall not be removed from the result as they usually are.
5872	0	For d, i, o, u, x, X, e, E, f, g, and G conversions, leading zeroes
5873		(following any indication of sign or base) shall be used to pad to
5874		the field width; no space padding shall be performed. If the 0 and
5875		– flags both appear, the 0 flag shall be ignored. For d, i, o, u, x,
5876		and X conversions, if a precision is specified, the 0 flag shall be
5877		ignored. For other conversions, the behavior is undefined.
5878	Each conversion character shall result in fetching zero or more arguments. The	
5879	results are undefined if there are insufficient arguments for the format. If the for-	
5880	mat is exhausted while arguments remain, the excess arguments shall be ignored.	
5881	The <i>conversion characters</i> and their meanings are:	
5882	d,i,o,u,x,X	The integer argument shall be written as signed decimal (d or i),
5883		unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal
5884		notation (x and X). The d and i specifiers shall convert to signed
5885		decimal in the style [-]dddd. The x conversion shall use the
5886		numbers and letters 0123456789abcdef and the X conversion
5887		shall use the numbers and letters 0123456789ABCDEF. The <i>preci-</i>
5888		<i>sion</i> component of the argument shall specify the minimum
5889		number of digits to appear. If the value being converted can be
5890		represented in fewer digits than the specified minimum, it shall
5891		be expanded with leading zeroes. The default precision shall be
5892		1. The result of converting a zero value with a precision of 0 shall
5893		be no characters. If both the field width and precision are omit-
5894		ted, the implementation may precede and/or follow numeric argu-
5895		ments of types d, i, and u with <blank>s; arguments of type o
5896		(octal) may be preceded with leading zeroes.
5897	f	The floating point number argument shall be written in decimal
5898		notation in the style "[-]ddd.ddd", where the number of digits
5899		after the radix character (shown here as a decimal point) shall be
5900		equal to the <i>precision</i> specification. The LC_NUMERIC locale
5901		category shall determine the radix character to use in this format.
5902		If the <i>precision</i> is omitted from the argument, six digits shall be
5903		written after the radix character; if the <i>precision</i> is explicitly 0, no
5904		radix character shall appear.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

5905	e,E	The floating point number argument shall be written in the style
5906		"[-]d.ddde±dd" (the symbol ± indicates either a plus or minus
5907		sign), where there is one digit before the radix character (shown
5908		here as a decimal point) and the number of digits after it is equal
5909		to the precision. The LC_NUMERIC locale category shall deter-
5910		mine the radix character to use in this format. When the preci-
5911		sion is missing, six digits shall be written after the radix charac-
5912		ter; if the precision is 0, no radix character shall appear. The E
5913		conversion character shall produce a number with E instead of e
5914		introducing the exponent. The exponent always shall contain at
5915		least two digits. However, if the value to be written requires an
5916		exponent greater than two digits, additional exponent digits shall
5917		be written as necessary.
5918	g,G	The floating point number argument shall be written in style f or
5919		e (or in style E in the case of a G conversion character), with the
5920		precision specifying the number of significant digits. The style
5921		used depends on the value converted: style e shall be used only if
5922		the exponent resulting from the conversion is less than -4 or
5923		greater than or equal to the precision. Trailing zeroes shall be
5924		removed from the result. A radix character shall appear only if it
5925		is followed by a digit.
5926	c	The integer argument shall be converted to an <i>unsigned char</i> and
5927		the resulting byte shall be written.
5928	s	The argument shall be taken to be a string and bytes from the
5929		string shall be written until the end of the string or the number of
5930		bytes indicated by the <i>precision</i> specification of the argument is
5931		reached. If the precision is omitted from the argument, it shall be
5932		taken to be infinite, so all bytes up to the end of the string shall
5933		be written.
5934	%	Write a % character; no argument shall be converted.
5935	In no case does a nonexistent or insufficient <i>field width</i> cause truncation of a field;	
5936	if the result of a conversion is wider than the field width, the field shall be simply	
5937	expanded to contain the conversion result.	

5938 **2.12.1 File Format Notation Rationale.** *(This subclause is not a part of P1003.2)*

5939 This clause was originally derived from the description of *printf()* in the *SVID*, but
 5940 it has been updated following the publication of the C Standard {7}. It is not
 5941 identical to the C Standard's {7} *printf()*, as it deals with integers as being essen-
 5942 tially one type, disregarding possible internal differences between *int*, *short*, and
 5943 *long*. It has also had some of the internal C language dependencies removed
 5944 (such as the requirement for null-terminated strings).

5945 This standard provides a rigorous description of the format of utility input and
 5946 output files. It is the intention of this standard that these descriptions be ade-
 5947 quate sources of information so that portable applications can use other utilities

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

such as `lex` or `awk` to reliably parse the output of these utilities as their input in, say a pipeline.

The notation for spaces allows some flexibility for application output. Note that an empty character position in *format* represents one or more `<blank>` characters on the output (not *white space*, which can include `<newline>`s). Therefore, another utility that reads that output as its input must be prepared to parse the data using `scanf()`, `awk`, etc. The `Δ` character is used when exactly one `<space>` is output.

The treatment of integers and spaces is different from the real `printf()`, in that they can be surrounded with `<blank>`s. This was done so that, given a format such as:

```
"%d\n", <foo>
```

the implementation could use a real `printf()` such as

```
printf("%6d\n", foo);
```

and still conform. It would have been possible for the standard to use `"%6d\n"`, but it would have been difficult to pick a number that would have pleased everyone. This notation is thus somewhat like `scanf()` in addition to `printf()`.

The `printf()` function was chosen as a model as most of the working group was familiar with it and it was thought that many of the readers would be as well.

One difference from the C function `printf()` is that the `l` and `h` conversion characters are not used. As expressed by this standard, there is no differentiation between decimal values for *ints* versus *longs* versus *shorts*. The specifications `%d` or `%i` should be interpreted as an arbitrary length sequence of digits. Also, no distinction is made between single precision and double precision numbers (*float/double* in C). These are simply referred to as floating point numbers.

Many of the output descriptions in this standard use the term *line*, such as:

```
"%s", <input line>
```

Since the definition of *line* includes the trailing `<newline>` character already, there is no need to include a `"\n"` in the format; a double `<newline>` would otherwise result.

In the language at the end of the clause:

```
"In no case does a nonexistent or insufficient field width cause truncation of a field; ..."
```

the term "field width" should not be confused with the term "precision" used in the description of `%s`.

Examples:

To represent the output of a program that prints a date and time in the form Sunday, July 3, 10:02, where `<weekday>` and `<month>` are strings:

```

5986      "%s, Δ%sΔ%d, Δ%d:%.2d\n", <weekday>, <month>, <day>, <hour>,
5987      <min>

5988  To show  $\pi$  written to 5 decimal places:

5989      "piΔ=Δ%.5f\n", <value of  $\pi$ >

5990  To show an input file format consisting of five colon-separated fields:

5991      "%s:%s:%s:%s:%s\n", <arg1>, <arg2>, <arg3>, <arg4>, <arg5>

```

2.13 Configuration Values

2.13.1 Symbolic Limits

This clause lists magnitude limitations imposed by a specific implementation. The braces notation, {LIMIT}, is used in this standard to indicate these values, but the braces are not part of the name.

Table 2-16 – Utility Limit Minimum Values

Name	Description	Value	
{POSIX2_BC_BASE_MAX}	The maximum <i>obase</i> value allowed by the <code>bc</code> utility.	99	
{POSIX2_BC_DIM_MAX}	The maximum number of elements permitted in an array by the <code>bc</code> utility.	2048	
{POSIX2_BC_SCALE_MAX}	The maximum <i>scale</i> value allowed by the <code>bc</code> utility.	99	
{POSIX2_BC_STRING_MAX}	The maximum length of a string constant accepted by the <code>bc</code> utility.	1000	
{POSIX2_COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the <code>LC_COLLATE</code> order keyword in the locale definition file; see 2.5.2.2.3.	2	
{POSIX2_EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the <code>expr</code> utility.	32	
{POSIX2_LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <code><newline></code> .	2048	
{POSIX2_RE_DUP_MAX}	The maximum number of repeated occurrences of a regular expression permitted when using the interval notation <code>\{m, n\}</code> ; see 2.8.3.3.	255	
{POSIX2_VERSION}	This value indicates the version of the utilities in this standard that are provided by the implementation. It will change with each published version of this standard.	199???	1
			1
			1
			1

The values specified in Table 2-16 represent the lowest values conforming implementations shall provide; and consequently, the largest values on which an application can rely without further enquiries, as described below. These values shall be accessible to applications via the `getconf` utility (see 4.26) and through the interfaces described in 7.8.2, [such as `sysconf()` in the C binding]. The literal names shown in the table apply only to the `getconf` utility; the high-level-language binding shall describe the exact form of each name to be used by the interfaces in that binding.

Implementations may provide more liberal, or less restrictive, values than shown in Table 2-16. These possibly more liberal values are accessible using the symbols in Table 2-17.

Table 2-17 – Symbolic Utility Limits

Name	Description	Minimum Value
{BC_BASE_MAX}	The maximum <i>obase</i> value allowed by the <code>bc</code> utility.	{POSIX2_BC_BASE_MAX}
{BC_DIM_MAX}	The maximum number of elements permitted in an array by the <code>bc</code> utility.	{POSIX2_BC_DIM_MAX}
{BC_SCALE_MAX}	The maximum <i>scale</i> value allowed by the <code>bc</code> utility.	{POSIX2_BC_SCALE_MAX}
{BC_STRING_MAX}	The maximum length of a string constant accepted by the <code>bc</code> utility.	{POSIX2_BC_STRING_MAX}
{COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the <code>LC_COLLATE</code> order keyword in the locale definition file; see 2.5.2.2.3.	{POSIX2_COLL_WEIGHTS_MAX}
{EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the <code>expr</code> utility.	{POSIX2_EXPR_NEST_MAX}
{LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of a utility's input line (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing <code><newline></code> .	{POSIX2_LINE_MAX}
{RE_DUP_MAX}	The maximum number of repeated occurrences of a regular expression permitted when using the interval notation <code>\{m,n\}</code> ; see 2.8.3.3.	{POSIX2_RE_DUP_MAX}

The functions in 7.8.2 [such as `sysconf()` in the C binding] or the `getconf` utility shall return the value of each symbol on each specific implementation. The value so retrieved shall be the largest, or most liberal, value that shall be available throughout the session lifetime, as determined at session creation. The literal names shown in the table apply only to the `getconf` utility; the high-level-language binding shall describe the exact form of each name to be used by the

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

interfaces in that binding.

All numerical limits defined by POSIX.1 {8}, such as {PATH_MAX}, also apply to this standard. (See POSIX.1 {8} 2.8.) All the utilities defined by this standard are implicitly limited by these values, unless otherwise noted in the utility descriptions.

It is not guaranteed that the application can in fact push a value to the implementation's specified limit in any given case, or at all, as a lack of virtual memory or other resources may prevent this. The limit value indicates only that the implementation does not specifically impose any arbitrary, more restrictive limit.

2.13.1.1 Symbolic Limits Rationale. *(This subclause is not a part of P1003.2)*

This clause grew out of an idea that originated in POSIX.1 {8}, in the form of `sysconf()` and `pathconf()`. (In fact, the same person wrote the original text for both standards.) The idea is that a Strictly Conforming POSIX.2 Application can be written to use the most restrictive values that a minimal system can provide, but it shouldn't have to. The values shown in Table 2-17 represent compromises so that some vendors can use historically-limited versions of UNIX system utilities. They are the highest values that Strictly Conforming POSIX.2 Applications or Conforming POSIX.2 Applications can assume, given no other information.

However, by using `getconf` or `sysconf()`, the elegant application can tailor itself to the more liberal values on some of the specific instances of specific implementations.

There is no explicitly-stated requirement that an implementation provide finite limits for any of these numeric values; the implementation is free to provide essentially unbounded capabilities (where it makes sense), stopping only at reasonable points such as {ULONG_MAX} (from the C Standard {7} via POSIX.1 {8}). Therefore, applications desiring to tailor themselves to the values on a particular implementation need to be ready for possibly huge values; it may not be a good idea to blindly allocate a buffer for an input line based on the value of {LINE_MAX}, for instance. However, unlike POSIX.1 {8}, there is no set of limits in this standard that return a special indication meaning "unbounded." The implementation should always return an actual number, even if the number is very large.

The statement

"It is not guaranteed that the application ...

is an indication that many of these limits are designed to ensure that implementors design their utilities without arbitrary constraints related to unimaginative programming. There are certainly conditions under which combinations of options can cause failures that would not render an implementation nonconforming. For example, {EXPR_NEST_MAX} and {ARG_MAX} could collide when expressions are large; combinations of {BC_SCALE_MAX} and {BC_DIM_MAX} could exceed virtual memory.

In POSIX.2, the notion of a limit being guaranteed for the process lifetime, as it is in POSIX.1 {8}, is not as useful to a shell script. The `getconf` utility is probably a process itself, so the guarantee would be valueless. Therefore, POSIX.2 requires the guarantee to be for the session lifetime. This will mean that many vendors will either return very conservative values or possibly implement `getconf` as a built-in.

It may seem confusing to have limits that apply only to a single utility grouped into one global clause. However, the alternative, which would be to disperse them out into their utility description clauses, would cause great difficulty when `sysconf()` and `getconf` were described. Therefore, the working group chose the global approach.

Each language binding could provide symbol names that are slightly different than are shown here. For example, the C binding prefixes the symbols with a leading underscore.

The following comments describe selection criteria for the symbols and their values.

{ARG_MAX}

This is defined by POSIX.1 {8}. Unfortunately, it is very difficult for a portable application to deal with this value, as it does not know how much of its argument space is being consumed by the user's environment variables.

{BC_BASE_MAX}

{BC_DIM_MAX}

{BC_SCALE_MAX}

These were originally one value, {BC_SCALE_MAX}, but it was unreasonable to link all three concepts into one limit.

{CHILD_MAX}

This is defined by POSIX.1 {8}.

{CUT_FIELD_MAX}

This value was removed from an earlier draft. It represented the maximum length of the *list* argument to the `cut -c` or `-f` options. Since the length is now unspecified, the utility should have to deal with arbitrarily long lists, as long as {ARG_MAX} is not exceeded.

{CUT_LINE_MAX}

This value was removed from an earlier draft. Historical cuts have had input line limits of 1024; this removal therefore mandates that a conforming `cut` shall process files with lines of unlimited length.

{DEPTH_MAX}

This directory-traversing depth limit (which at one time applied to `rm` and `find`) was removed from an earlier draft for two major reasons:

- (1) It could be a security problem if utilities searching for files could not descend below a published depth; this would be a semi-reliable means of hiding files from the administrator.

6157 (2) There is no reason a reasonable implementation should have to
6158 limit itself in this way.

6159 {ED_FILE_MAX}
6160 This value was removed from an earlier draft. Historical eds have had
6161 very small file limits; since {ED_FILE_MAX} is no longer specified, imple-
6162 mentations have to document the limits as described in 2.11. It is
6163 recommended that implementations set much more reasonable file size
6164 limits as they modify ed to deal with other features required by POSIX.2.

6165 {ED_LINE_MAX}
6166 This value was removed from an earlier draft. Historical eds have had
6167 small input line limits; this removal therefore mandates that a conform-
6168 ing ed shall process files with lines of length {LINE_MAX}.

6169 {COLL_WEIGHTS_MAX}
6170 The weights assigned to order can be considered as “passes” through
6171 the collation algorithm.

6172 {EXPR_NEST_MAX}
6173 The value for expression nesting was borrowed from the C Standard {7}.

6174 {FIND_DEPTH_MAX}
6175 This was removed from an earlier draft in favor of a common value,
6176 {DEPTH_MAX}.

6177 {FIND_FILESYS_MAX}
6178 This was removed from an earlier draft. It indicated the limit of the
6179 number of file systems that find could traverse in its search. It was
6180 dropped because this standard does not really acknowledge the histori-
6181 cal nature of separate file systems.

6182 {FIND_NEWER_MAX}
6183 This value, which allowed find to limit the number of -newer operands
6184 it processed, was deleted from an earlier draft. It was felt to be a ves-
6185 tige of a particular implementation with an incorrect programming algo-
6186 rithm that should not limit applications.

6187 {JOIN_LINE_MAX}
6188 This value was removed from an earlier draft. Historical joins have
6189 had input line limits of 1024; this removal therefore mandates that a
6190 conforming join shall process files with lines of length {LINE_MAX}.

6191 {LINE_MAX}
6192 This is a global limit that affects all utilities, unless otherwise noted.
6193 The {MAX_CANON} value from POSIX.1 {8} may further limit input lines
6194 from terminals. The {LINE_MAX} value was the subject of much debate
6195 and is a compromise between those who wished unlimited lines and
6196 those who understood that many historical utilities were written with
6197 fixed buffers. Frequently, utility writers selected the UNIX system con-
6198 stant BUFSIZ to allocate these buffers; therefore, some utilities were
6199 limited to 512 bytes for I/O lines, while others achieved 4096 or greater.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

6200 It should be noted that {LINE_MAX} applies only to input line length;
 6201 there is no requirement in the standard that limits the length of output
 6202 lines. Utilities such as `awk`, `sed`, and `paste` could theoretically con-
 6203 struct lines longer than any of the input lines they received, depending
 6204 on the options used or the instructions from the application. They are
 6205 not required to truncate their output to {LINE_MAX}. It is the responsi-
 6206 bility of the application to deal with this. If the output of one of those
 6207 utilities is to be piped into another of the standard utilities, line lengths
 6208 restrictions will have to be considered; the `fold` utility, among others,
 6209 could be used to ensure that only reasonable line lengths reach utilities
 6210 or applications.

6211 {LINK_MAX}

6212 This is defined by POSIX.1 {8}.

6213 {LP_LINE_MAX}

6214 This value was removed from an earlier draft. Since so little is being
 6215 required for the details of the `lp` utility, it made little sense to specify
 6216 how long its output lines are. Thus, implementations of `lp` will be
 6217 expected to deal with lines up to {LINE_MAX}, but whether those lines
 6218 print sensibly on every device is unspecified.

6219 {MAX_CANON}

6220 This is defined by POSIX.1 {8}.

6221 {MAX_INPUT}

6222 This is defined by POSIX.1 {8}.

6223 {NAME_MAX}

6224 This is defined by POSIX.1 {8}.

6225 {NGROUPS_MAX}

6226 This is defined by POSIX.1 {8}.

6227 {OPEN_MAX}

6228 This is defined by POSIX.1 {8}.

6229 {PATH_MAX}

6230 This is defined by POSIX.1 {8}.

6231 {PIPE_BUF}

6232 This is defined by POSIX.1 {8}.

6233 {RM_DEPTH_MAX}

6234 This was removed from an earlier draft in favor of a common value,
 6235 {DEPTH_MAX}.

6236 {RE_DUP_MAX}

6237 The value selected is consistent with historical practice.

6238 {SED_PATTERN_MAX}

6239 This symbolic value, the size of the `sed` pattern space, was replaced by a
 6240 specific value in the `sed` description. It is unlikely that any real appli-
 6241 cation would ever need to access this value symbolically.

{SORT_LINE_MAX}

This was removed from an earlier draft. Now that `cut` and `fold` can handle unlimited-length input lines, a special long input line limit for `sort` is not needed.

There are different limits associated with command lines and input to utilities, depending on the method of invocation. In the case of a C program *exec*-ing a utility, {ARG_MAX} is the underlying limit. In the case of the shell reading a script and *exec*-ing a utility, {LINE_MAX} limits the length of lines the shell is required to process and {ARG_MAX} will still be a limit. If a user is entering a command on a terminal to the shell, requesting that it invoke the utility, {MAX_INPUT} may restrict the length of the line that can be given to the shell to a value below {LINE_MAX}.

2.13.2 Symbolic Constants for Portability Specifications

Table 2-18 – Optional Facility Configuration Values

Name	Description
{POSIX2_C_BIND}	The C language development facilities in Annex A support the C Language Bindings Option (see Annex B).
{POSIX2_C_DEV}	The system supports the C Language Development Utilities Option (see Annex A).
{POSIX2_FORT_DEV}	The system supports the FORTRAN Development Utilities Option (see Annex C).
{POSIX2_FORT_RUN}	The system supports the FORTRAN Runtime Utilities Option (see Annex C).
{POSIX2_LOCALEDEF}	The system supports the creation of locales as described in 4.35.
{POSIX2_SW_DEV}	The system supports the Software Development Utilities Option (see Section 6).

Table 2-18 lists symbols that can be used by the application to determine which optional facilities are present on the implementation. The functions defined in 7.8.2 [such as *sysconf()*] or the *getconf* utility can be used to retrieve the value of each symbol on each specific implementation. The literal names shown in the table apply only to the *getconf* utility; the high-level-language binding shall describe the exact form of each name to be used by the interfaces in that binding.

Each of these symbols shall be considered valid names by the implementation. Each shall be defined on the system with a value of 1 if the corresponding option is supported; otherwise, the symbol shall be undefined.

2.13.2.1 Symbolic Constants for Portability Specifications Rationale. *(This subclause is not a part of P1003.2)*

When an option is supported, `getconf` returns a value of 1. For example, when C development is supported:

```
if [ "$(getconf POSIX2_C_DEV)" -eq 1 ]; then
    echo C supported
fi
```

The `sysconf()` function in the C binding would return 1.

The following comments describe selection criteria for the symbols and their values.

```
{POSIX2_C_BIND}
{POSIX2_C_DEV}
{POSIX2_FORT_DEV}
{POSIX2_SW_DEV}
```

These were renamed from `_POSIX_*` in Draft 9 after it was pointed out that each of the POSIX standards should keep generally in its own namespace.

It is possible for some (usually privileged) operations to remove utilities that support these options, or otherwise render these options unsupported. The header files, the `sysconf()` function, or the `getconf` utility will not necessarily detect such actions, in which case they should not be considered as rendering the implementation nonconforming. A test suite should not attempt tests like:

```
rm /usr/bin/c89
getconf POSIX2_C_DEV
```

```
{_POSIX_LOCALEDEF}
```

This symbol was introduced to allow implementations to restrict supported locales to only those supplied by the implementation.

Section 3: Shell Command Language

The shell is a command language interpreter. This section describes the syntax of that command language as it is used by the `sh` utility and the functions in 7.1 [such as `system()` and `popen()` in the C binding].

The shell operates according to the following general overview of operations. The specific details are included in the cited clauses and subclauses of this section. The shell:

- (1) Reads its input from a file (see `sh` in 4.56), from the `-c` option, or from one of the functions in 7.1. If the first line of a file of shell commands starts with the characters `#!`, the results are unspecified.
- (2) Breaks the input into tokens: words and operators. (See 3.3.)
- (3) Parses the input into simple (3.9.1) and compound (3.9.4) commands.
- (4) Performs various expansions (separately) on different parts of each command, resulting in a list of pathnames and fields to be treated as a command and arguments (3.6).
- (5) Performs redirection (3.7) and removes redirection operators and their operands from the parameter list.
- (6) Executes a function (3.9.5), built-in (3.14), executable file, or script, giving the name of the command (or, in the case of a function within a script, the name of the script) as the “zero’t” argument and the remaining words and fields as parameters (3.9.1.1).
- (7) Optionally waits for the command to complete and collects the exit status (3.8.2).

3.0.1 Shell Command Language Rationale. *(This subclause is not a part of P1003.2)*

The System V shell was selected as the starting point for this standard. The BSD C-shell was excluded from consideration, for the following reasons:

- (1) Most historically portable shell scripts assume the Version 7 “Bourne” shell, from which the System V shell is derived.
- (2) The majority of tutorial materials on shell programming assume the System V shell.

Despite the selection of the System V shell, the developers of the standard did not limit the possibilities for a shell command language that was upward-compatible.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

The only programmatic interfaces to the shell language are through the functions in 7.1 and the `sh` utility. Most implementations provide an interface to, and processing mode for, the shell that is suitable for direct user interaction. The behavior of this interactive mode is not defined by this standard; however, places where historically an interactive shell behaves differently from the behavior described here are noted.

- (1) Aliases are not included in the base POSIX.2 because they duplicate functionality already available to applications with functions. In early drafts, the search order of simple command lookup was “aliases, built-ins, functions, file system,” and therefore an alias was necessary to create a user-defined command having the same name as a built-in. To retain this capability, the search order has changed to “special built-ins, functions, built-ins, file system,” and a built-in, called `command`, has been added, which disables the looking up of functions. Aliases are a part of the POSIX.2a UPE because they are widely used by human users, as differentiated from applications.
- (2) All references to job control and related commands have been omitted from the base POSIX.2. POSIX.2 describes the noninteractive operation of the shell; job control is outside the scope of this standard until the UPE revision is developed. Apparently it is not widely known that traditionally, even in a job control environment, the commands executed during the execution of a shell script are not placed into separate process groups. If they were, one could not stop the execution of the shell script from the interactive shell, for example. This standard does not require or prohibit job control; it simply does not mention it.
- (3) The conditional command (double bracket `[[]]`) was removed from an earlier draft. Objections were lodged that the real problem is misuse of the `test` command (`()`), and putting it into the shell is the wrong way to fix the problem. Instead, proper documentation and a new shell reserved word (`!`) are sufficient. Tests that require multiple `test` operations can be done at the shell level using individual invocations of the `test` command and shell logicals, rather than the error prone `-o` flag of `test`.
- (4) Exportable functions were removed from an earlier draft. See the rationale in 3.9.5.1.

The construct `#!` is reserved for implementations wishing to provide that extension. If it were not reserved, the standard would disallow it by forcing it to be a comment. As it stands, a conforming application shall not use `#!` as the first line of a shell script.

3.1 Shell Definitions

The following terms are used in Section 3. Because they are specific to the shell, they do not appear in 2.2.2.

3.1.1 control operator: A token that performs a control function.

It is one of the following symbols:

&)	<newline>
&&	;	
(;;	

The end-of-input indicator used internally by the shell is also considered a control operator. See 3.3.

On some systems, the symbol ((is a control operator; its use produces unspecified results.

3.1.2 expand: When not qualified, the act of applying all the expansions described in 3.6.

3.1.3 field: A unit of text that is the result of parameter expansion (3.6.2), arithmetic expansion (3.6.4), command substitution (3.6.3), or field splitting (3.6.5).

During command processing (see 3.9.1), the resulting fields are used as the command name and its arguments.

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

The behavior in this mode is not defined by this standard.

NOTE: The preceding sentence is expected to change following the eventual approval of the UPE supplement.

3.1.5 name: A word consisting solely of underscores, digits, and alphabets from the portable character set (see 2.4).

The first character of a name shall not be a digit.

3.1.6 operator: Either a control operator or a redirection operator.

3.1.7 parameter: An entity that stores values.

There are three types of parameters: variables (named parameters), positional parameters, and special parameters. Parameter expansion is accomplished by introducing a parameter with the \$ character. See 3.5.

3.1.8 positional parameter: A parameter denoted by a single digit or one or more digits in curly braces.

See 3.5.1.

3.1.9 redirection: A method of associating files with the input/output of commands.

See 3.7.

3.1.10 redirection operator: A token that performs a redirection function.

It is one of the following symbols:

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

3.1.11 special parameter: A parameter named by a single character from the following list:

* @ # ? ! - \$ 0

See 3.5.2.

3.1.12 subshell: A shell execution environment, distinguished from the main or current shell execution environment by the attributes described in 3.12.

3.1.13 token: A sequence of characters that the shell considers as a single unit when reading input, according to the rules in 3.3.

A token is either an operator or a word.

3.1.14 variable: A named parameter. See 3.5.

3.1.15 variable assignment [assignment]: A word consisting of the following parts

varname=value

When used in a context where assignment is defined to occur (see 3.9.1) and at no other time, the *value* (representing a word or field) shall be assigned as the value of the variable denoted by *varname*. The *varname* and *value* parts meet the requirements for a name and a word, respectively, except that they are delimited by the embedded unquoted equals-sign in addition to the delimiting described in 3.3. In all cases, the variable shall be created if it did not already exist. If *value* is not specified, the variable shall be given a null value.

An alternative form of variable assignment:

symbol=value

(where *symbol* is a valid word delimited by an equals-sign, but not a valid name) produces unspecified results.

3.1.16 word: A token other than an operator.

In some cases a word is also a portion of a word token: in the various forms of parameter expansion (3.6.2), such as $\${name-word}$, and variable assignment, such as *name=word*, the word is the portion of the token depicted by *word*. The concept of a word is no longer applicable following word expansions—only fields remain; see 3.6.

3.1.17 Shell Definitions Rationale. *(This subclause is not a part of P1003.2)*

The *word=word* form of variable assignment was included, producing unspecified results, to allow the KornShell *name[expression]=value* syntax to conform.

The `((` symbol is a control operator in the KornShell, used for an alternative syntax of an arithmetic expression command. A strictly conforming POSIX.2 application cannot use `((` as a single token [with the obvious exception of the `$((` form described in POSIX.2]. The decision to require this is based solely on the pragmatic knowledge that there are many more historical shell scripts using the KornShell syntax than there might be using nested subshells, such as

```
((foo))      or      ((foo);(bar))
```

The latter example should not be misinterpreted by the shell as arithmetic because attempts to balance the parentheses pairs would indicate that they are subshells. Thus, in most cases, while a few scripts will no longer be strictly portable, the chances of breaking existing scripts is even smaller.

There are no explicit limits in this standard on the sizes of names, words, lines, or other objects. However, other implicit limits do apply: shell script lines produced by many of the standard utilities cannot exceed `{LINE_MAX}` and the sum of exported variables comes under the `{ARG_MAX}` limit. Historical shells dynamically allocate memory for names and words and parse incoming lines a byte at a time. Lines cannot have an arbitrary `{LINE_MAX}` limit because of historical practice such as `makefiles`, where `make` removes the `<newline>`s associated with the commands for a target and presents the shell with one very long line. The text in 2.11.5.2 does allow a shell to run out of memory, but it cannot have arbitrary programming limits.

3.2 Quoting

Quoting is used to remove the special meaning of certain characters or words to the shell. Quoting can be used to preserve the literal meaning of the special characters in the next paragraph; prevent reserved words from being recognized as such; and prevent parameter expansion and command substitution within here-document processing (see 3.7.4).

The following characters shall be quoted if they are to represent themselves:

```
|   &   ;   <   >   (   )   $   '   \   "   '
<space>   <tab>   <newline>
```

and the following may need to be quoted under certain circumstances. That is, these characters may be special depending on conditions described elsewhere in the standard:

```
*   ?   [   #   ~   =   %
```

The various quoting mechanisms are the escape character, single-quotes, and double-quotes. The here-document represents another form of quoting; see 3.7.4.

3.2.1 Escape Character (Backslash)

A backslash that is not quoted shall preserve the literal value of the following character, with the exception of a <newline>. If a <newline> follows the backslash, the shell shall interpret this as line continuation. The backslash and <newline> shall be removed before splitting the input into tokens.

3.2.2 Single-Quotes

Enclosing characters in single-quotes (' ') shall preserve the literal value of each character within the single-quotes. A single-quote cannot occur within single-quotes.

3.2.3 Double-Quotes

Enclosing characters in double-quotes (" ") shall preserve the literal value of all characters within the double-quotes, with the exception of the characters dollar-sign, backquote, and backslash, as follows:

\$ The dollar-sign shall retain its special meaning introducing parameter expansion (see 3.6.2), a form of command substitution (see 3.6.3), and arithmetic expansion (see 3.6.4).

The input characters within the quoted string that are also enclosed between \$(and the matching) shall not be affected by the double-quotes, but rather shall define that command whose output replaces the \$(...) when the word is expanded. The tokenizing rules in 3.3 shall be applied recursively to find the matching).

Within the string of characters from an enclosed \${ to the matching }, an even number of unescaped double-quotes or single-quotes, if any, shall occur. A preceding backslash character shall be used to escape a literal { or }. The rule in 3.6.2 shall be used to determine the matching }.

` The backquote shall retain its special meaning introducing the other form of command substitution (see 3.6.3). The portion of the quoted string from the initial backquote and the characters up to the next backquote that is not preceded by a backslash, having escape characters removed, defines that command whose output replaces `...` when the word is expanded. Either of the following cases produces undefined results:

— A single- or double-quoted string that begins, but does not end, within the `...` sequence.

— A `...` sequence that begins, but does not end, within the same double-quoted string.

216 \ The backslash shall retain its special meaning as an escape character
 217 (see 3.2.1) only when followed by one of the characters:

218 \$ ' " \ <newline>

219 A double-quote shall be preceded by a backslash to be included within double-
 220 quotes. The parameter @ has special meaning inside double-quotes and is
 221 described in 3.5.2.

222 **3.2.4 Quotes Rationale.** *(This subclause is not a part of P1003.2)*

223 A backslash cannot be used to escape a single-quote in a single-quoted string. An
 224 embedded quote can be created by writing, for example, 'a'\''b', which yields
 225 a'b. (See 3.6.5 for a better understanding of how portions of words are either
 226 split into fields or remain concatenated.) A single token can be made up of con-
 227 catenated partial strings containing all three kinds of quoting/escaping, thus per-
 228 mitting any combination of characters.

229 The escaped <newline> used for line continuation is removed entirely from the
 230 input and is not replaced by any white space. Therefore, it cannot serve as a
 231 token separator.

232 In double-quoting, if a backslash is immediately followed by a character that
 233 would be interpreted as having a special meaning, the backslash is deleted and
 234 the subsequent character is taken literally. If a backslash does not precede a
 235 character that would have a special meaning, it is left in place unmodified and the
 236 character immediately following it is also left unmodified. Thus, for example:

237 "\\$" ⇒ \$

238 "\a" ⇒ \a

239 It would be desirable to include the statement "The characters from an enclosed
 240 \${ to the matching } shall not be affected by the double-quotes," similar to the
 241 one for \$(). However, historical practice in the System V shell prevents this.
 242 The requirement that double-quotes be matched inside \${...} within double-
 243 quotes and the rule for finding the matching } in 3.6.2 eliminate several subtle
 244 inconsistencies in expansion for historical shells in rare cases; for example,

245 "\${foo-bar}"

246 yields bar when foo is not defined, and is an invalid substitution when foo is 1
 247 defined, in many historical shells. The differences in processing the "\${...}"
 248 form have led to inconsistencies between the historical System V, BSD, and Korn-
 249 Shells, and the text in POSIX.2 is an attempt to converge them without breaking
 250 many applications. A consequence of the new rule is that single-quotes cannot be
 251 used to quote the } within "\${...}"; for example

252 unset bar
 253 foo="\${bar-' }' }

254 is invalid because the "\${...}" substitution contains an unpaired unescaped 1
 255 single-quote. The backslash can be used to escape the } in this example to 1

achieve the desired result:

```
unset bar
foo="${bar-\\}"
```

The only alternative to this compromise between shells would be to make the behavior unspecified whenever the literal characters `'`, `{`, `}`, and `"` appear within `${...}`. To write a portable script that uses these values, a user would have to assign variables, say,

```
squote='\ dquote=\" lbrace='{ ' rbrace='}'
${foo-$squote$rbrace$squote}
```

rather than

```
${foo-"'"}
```

Some systems have allowed the end of the word to terminate the backquoted command substitution, such as in

```
"`echo hello"
```

This usage is undefined in POSIX.2, where the matching backquote is required. The other undefined usage can be illustrated by the example:

```
sh -c `` echo "foo``
```

The description of the recursive actions involving command substitution can be illustrated with an example. Upon recognizing the introduction of command substitution, the shell must parse input (in a new context), gathering the “source” for the command substitution until an unbalanced `)` or ``` is located. For example, in the following

```
echo "${date; echo "
    one" )"
```

the double-quote following the `echo` does not terminate the first double-quote; it is part of the command substitution “script.” Similarly, in

```
echo "${echo *}"
```

the asterisk is not quoted since it is inside command substitution; however,

```
echo "${echo "*"}
```

is quoted (and represents the asterisk character itself).

3.3 Token Recognition

The shell reads its input in terms of lines from a file, from a terminal in the case of an interactive shell, or from a string in the case of `sh -c` or `system()`. The input lines can be of unlimited length. These lines are parsed using two major modes: ordinary token recognition and processing of here-documents.

When an `io_here` token has been recognized by the grammar (see 3.10), one or more of the immediately subsequent lines form the body of one or more here-

documents and shall be parsed according to the rules of 3.7.4.

When it is not processing an `io_here`, the shell shall break its input into tokens by applying the first applicable rule below to the next character in its input. The token shall be from the current position in the input until a token is delimited according to one of the rules below; the characters forming the token are exactly those in the input, including any quoting characters. If it is indicated that a token is delimited, and no characters have been included in a token, processing shall continue until an actual token is delimited.

- (1) If the end of input is recognized, the current token shall be delimited. If there is no current token, the end-of-input indicator shall be returned as the token.
- (2) If the previous character was used as part of an operator and the current character is not quoted and can be used with the current characters to form an operator, it shall be used as part of that (operator) token.
- (3) If the previous character was used as part of an operator and the current character cannot be used with the current characters to form an operator, the operator containing the previous character shall be delimited.
- (4) If the current character is backslash, single-quote, or double-quote (`\`, `'`, or `"`) and it is not quoted, it shall affect quoting for subsequent character(s) up to the end of the quoted text. The rules for quoting are as described in 3.2. During token recognition no substitutions shall be actually performed, and the result token shall contain exactly the characters that appear in the input (except for `<newline>` joining), unmodified, including any embedded or enclosing quotes or substitution operators, between the quote mark and the end of the quoted text. The token shall not be delimited by the end of the quoted field.
- (5) If the current character is an unquoted `$` or ```, the shell shall identify the start of any candidates for parameter expansion (3.6.2), command substitution (3.6.3), or arithmetic expansion (3.6.4) from their introductory unquoted character sequences: `$` or `${`, `$ (` or ```, and `$ ((`, respectively. The shell shall read sufficient input to determine the end of the unit to be expanded (as explained in the cited subclauses). While processing the characters, if instances of expansions or quoting are found nested within the substitution, the shell shall recursively process them in the manner specified for the construct that is found. The characters found from the beginning of the substitution to its end, allowing for any recursion necessary to recognize embedded constructs, shall be included unmodified in the result token, including any embedded or enclosing substitution operators or quotes. The token shall not be delimited by the end of the substitution.
- (6) If the current character is not quoted and can be used as the first character of a new operator, the current token (if any) shall be delimited. The current character shall be used as the beginning of the next (operator) token.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

- (7) If the current character is an unquoted <newline>, the current token shall be delimited.
- (8) If the current character is an unquoted <blank>, any token containing the previous character is delimited and the current character is discarded.
- (9) If the previous character was part of a word, the current character is appended to that word.
- (10) If the current character is a #, it and all subsequent characters up to, but excluding, the next <newline> are discarded as a comment. The <newline> that ends the line is not considered part of the comment.
- (11) The current character is used as the start of a new word.

Once a token is delimited, it shall be categorized as required by the grammar in 3.10.

3.3.1 Token Recognition Rationale. *(This subclause is not a part of P1003.2)*

The (3) rule about combining characters to form operators is not meant to preclude systems from extending the shell language when characters are combined in otherwise invalid ways. Portable applications cannot use invalid combinations and test suites should not penalize systems that take advantage of this fact. For example, the unquoted combination |& is not valid in a POSIX.2 script, but has a specific KornShell meaning.

The (10) rule about # as the current character is the first in the sequence in which a new token is being assembled. The # starts a comment only when it is at the beginning of a token. This rule is also written to indicate that the search for the end-of-comment does not consider escaped <newline> specially, so that a comment cannot be continued to the next line.

3.4 Reserved Words

Reserved words are words that have special meaning to the shell. (See 3.9.) The following words shall be recognized as reserved words:

!	elif	fi	in	while
case	else	for	then	{ ⁴⁾
do	esac	if	until	}
done				

4) In some historical systems, the curly braces are treated as control operators. To assist in future standardization activities, portable applications should avoid using unquoted braces to represent the characters themselves. It is possible that a future version of POSIX.2 may require this, although probably not for the often-used `find { }` construct.

This recognition shall occur only when none of the characters are quoted and when the word is used as:

- (1) The first word of a command
- (2) The first word following one of the reserved words other than `case`, `for`, or `in`
- (3) The third word in a `case` or `for` command (only `in` is valid in this case)

See the grammar in 3.10.

The following words may be recognized as reserved words on some systems (when none of the characters are quoted), causing unspecified results:

`function` `select` `[[` `]]`

2

Words that are the concatenation of a name and a colon (`:`) are reserved; their use produces unspecified results.

3.4.1 Reserved Words Rationale. *(This subclause is not a part of P1003.2)*

All reserved words are recognized syntactically as such in the contexts described. However, it is useful to point out that `in` is the only meaningful reserved word after a `case` or `for`; similarly, `in` is not meaningful as the first word of a simple command.

Reserved words are recognized only when they are delimited (i.e., meet the definition of *word*; see 3.1.16), whereas operators are themselves delimiters. For instance, `(` and `)` are control operators, so that no `<space>` is needed in `(list)`. However, `{` and `}` are reserved words in `{ list; }`, so that in this case the leading `<space>` and semicolon are required.

The list of unspecified reserved words is from the KornShell, so portable applications cannot use them in places a reserved word would be recognized. This list contained `time` in earlier drafts, but it was removed when the `time` utility was selected for the UPE.

There was a strong argument for promoting braces to operators (instead of reserved words), so they would be syntactically equivalent to subshell operators. Concerns about compatibility outweighed the advantages of this approach. Nevertheless, portable applications should consider quoting `{` and `}` when they represent themselves.

The restriction on ending a name with a colon is to allow future implementations that support named labels for flow control. See the rationale for `break` (3.14.1.1).

3.5 Parameters and Variables

A parameter can be denoted by a name, a number, or one of the special characters listed in 3.5.2. A variable is a parameter denoted by a name.

A parameter is set if it has an assigned value (null is a valid value). Once a variable is set, it can only be unset by using the `unset` special built-in command.

3.5.1 Positional Parameters

A positional parameter is a parameter denoted by the decimal value represented by one or more digits, other than the single digit 0. When a positional parameter with more than one digit is specified, the application shall enclose the digits in braces (see 3.6.2). Positional parameters are initially assigned when the shell is invoked (see `sh` in 4.56), temporarily replaced when a shell function is invoked (see 3.9.5), and can be reassigned with the `set` special built-in command.

3.5.1.1 Positional Parameters Rationale. *(This subclause is not a part of P1003.2)*

The digits denoting the positional parameters are always interpreted as a decimal value, even if there is a leading zero.

3.5.2 Special Parameters

Listed below are the special parameters and the values to which they shall expand. Only the values of the special parameters are listed; see 3.6 for a detailed summary of all the stages involved in expanding words.

- * Expands to the positional parameters, starting from one. When the expansion occurs within a double-quoted string (see 3.2.3), it expands to a single field with the value of each parameter separated by the first character of the `IFS` variable, or by a `<space>` if `IFS` is unset.
- @ Expands to the positional parameters, starting from one. When the expansion occurs within double-quotes, each positional parameter expands as a separate field, with the provision that the expansion of the first parameter is still joined with the beginning part of the original word (assuming that the expanded parameter was embedded within a word), and the expansion of the last parameter is still joined with the last part of the original word. If there are no positional parameters, the expansion of @ shall generate zero fields, even when @ is double-quoted.
- # Expands to the decimal number of positional parameters.
- ? Expands to the decimal exit status of the most recent pipeline (see 3.9.2).
- (Hyphen) Expands to the current option flags (the single-letter option names concatenated into a string) as specified on invocation, by the `set` special built-in command, or implicitly by the shell.

443 \$ Expands to the decimal process ID of the invoked shell. In a subshell
 444 (see 3.12), \$ shall expand to the same value as that of the current shell.

445 ! Expands to the decimal process ID of the most recent background com-
 446 mand (see 3.9.3) executed from the current shell. For a pipeline, the 1
 447 process ID is that of the last command in the pipeline.

448 0 (Zero.) Expands to the name of the shell or shell script. See sh (4.56)
 449 for a detailed description of how this name is derived.

450 See the description of the **IFS** variable in 3.5.3.

451 **3.5.2.1 Special Parameters Rationale.** *(This subclause is not a part of P1003.2)*

452 Most historical implementations implement subshells by forking; thus, the special
 453 parameter \$ does not necessarily represent the process ID of the shell process exe-
 454 cuting the commands since the subshell execution environment preserves the
 455 value of \$.

456 If a subshell were to execute a background command, the value of its parent's \$! 1
 457 would not change. For example: 1

```
458         ( 1
459         date & 1
460         echo $! 1
461         ) 1
462         echo $! 1
```

463 would echo two different values for \$!. 1

464 The descriptions of parameters * and @ assume the reader is familiar with the
 465 field splitting discussion in 3.6.5 and understands that portions of the word will
 466 remain concatenated unless there is some reason to split them into separate
 467 fields. Some examples of the * and @ properties, including the concatenation
 468 aspects:

```
469         set "abc" "def ghi" "jkl"
470         echo $*      => "abc" "def" "ghi" "jkl"
471         echo "$*"    => "abc def ghi jkl"
472         echo $@      => "abc" "def" "ghi" "jkl"
```

473 *but*

```
474         echo "$@"    => "abc" "def ghi" "jkl"
475         echo "xx$@yy" => "xxabc" "def ghi" "jkl"
476         echo "$@$@"  => "abc" "def ghi" "jklabc" "def ghi" "jkl"
```

477 In the preceding examples, the double-quote characters that appear after the =>
 478 do not appear in the output and are used only to illustrate word boundaries.

479 Historical versions of the Bourne shell have used <space> as a separator
 480 between the expanded members of "\$*". The KornShell has used the first char-
 481 acter in **IFS**, which is <space> by default. If **IFS** is set to a null string, this is not 1
 482 equivalent to unsetting it; its first character will not exist, so the parameter 1
 483 values are concatenated. For example: 1

```

484      $ IFS=' '                                1
485      $ set foo bar bam                        1
486      $ echo "$@"                             1
487      foo bar bam                             1
488      $ echo "$*"                             1
489      foobarbam                               1
490      $ unset IFS                             1
491      $ echo "$*"                             1
492      foo bar bam                             1

493  The $- can be used to save and restore set options:

494      Save=$(echo $- | sed 's/[ics]//g')        1
495      ...
496      set +aCefnuvx                             2
497      set -$Save

498  The three options are removed using sed in the example because they may 1
499  appear in the value of $- (from the sh command line), but are not valid options to 1
500  set.                                           1

501  The command name (parameter 0) is not counted in the number given by #
502  because it is a special parameter, not a positional parameter.

```

503 3.5.3 Variables

504 Variables shall be initialized from the environment (as defined by POSIX.1 {8})
 505 and can be given new values with variable assignment commands. If a variable is
 506 initialized from the environment, it shall be marked for export immediately; see
 507 3.14.8. New variables can be defined and initialized with variable assignments,
 508 with the `read` or `getopts` utilities, with the *name* parameter in a `for` loop (see
 509 3.9.4.2), with the `${name=word}` expansion, or with other mechanisms provided
 510 as implementation extensions. The following variables shall affect the execution
 511 of the shell:

512	HOME	This variable shall be interpreted as the pathname of the user's home directory. The contents of HOME are used in Tilde Expansion (see 3.6.1).
513		
514		
515	IFS	<i>Input field separators</i> : a string treated as a list of characters that is used for field splitting and to split lines into fields with the <code>read</code> command. If IFS is not set, the shell shall behave as if the value of IFS were the <space>, <tab>, and <newline> characters. (See 3.6.5.)
516		
517		
518		
519		
520	LANG	This variable shall provide a default value for the LC_* variables, as described in 2.6.
521		
522	LC_ALL	This variable shall interact with the LANG and LC_* variables as described in 2.6.
523		

524	LC_COLLATE	This variable shall determine the behavior of range	
525		expressions, equivalence classes, and multicharacter col-	
526		lating elements within pattern matching.	
527	LC_CTYPE	This variable shall determine the interpretation of	
528		sequences of bytes of text data as characters (e.g., single-	
529		versus multibyte characters), which characters are	
530		defined as letters (character class <code>alpha</code>), and the	
531		behavior of character classes within pattern matching.	
532	LC_MESSAGES	This variable shall determine the language in which mes-	
533		sages should be written.	
534	PATH	This variable represents a string formatted as described	
535		in 2.6, used to effect command interpretation. See 3.9.1.1.	1

536 **3.5.3.1 Variables Rationale.** *(This subclause is not a part of P1003.2)*

537 A description of **PWD** (which is automatically set by the KornShell whenever the
538 current working directory changes) was omitted because its functionality is easily
539 reproduced using `$(pwd)`.

540 See the discussion of **IFS** in 3.6.5.1.

541 Other common environment variables used by historical shells are not specified
542 by this standard, but they should be reserved for the historical uses. For interac-
543 tive use, other shell variables are expected to be introduced by the UPE (and this
544 rationale will be updated accordingly): **ENV**, **FCEDIT**, **HISTFILE**, **HISTSIZ**,
545 **LINENO**, **PPID**, **PS1**, **PS2**, **PS4**.

546 Tilde expansion for components of the **PATH** in an assignment such as:

547 `PATH=~hlj/bin:~dwc/bin:$PATH` 1

548 is a feature of some historical shells and is allowed by the wording of 3.6.1. Note 1
549 that the tildes are expanded during the assignment to **PATH**, not when **PATH** is 1
550 accessed during command search. 1

551 **3.6 Word Expansions**

552 This clause describes the various expansions that are performed on words. Not
553 all expansions are performed on every word, as explained in the following sub-
554 clauses.

555 Tilde expansions, parameter expansions, command substitutions, arithmetic
556 expansions, and quote removals that occur within a single word expand to a sin-
557 gle field. It is only field splitting or pathname expansion that can create multiple
558 fields from a single word. The single exception to this rule is the expansion of the
559 special parameter `@` within double-quotes, as is described in 3.5.2.

560 The order of word expansion shall be as follows:

- (1) Tilde Expansion (see 3.6.1), Parameter Expansion (see 3.6.2), Command Substitution (see 3.6.3), and Arithmetic Expansion (see 3.6.4) shall be performed, beginning to end. [See item (5) in 3.3.]
- (2) Field Splitting (see 3.6.5) shall be performed on fields generated by step (1) unless **IFS** is null.
- (3) Pathname Expansion (see 3.6.6) shall be performed, unless `set -f` is in effect.
- (4) Quote Removal (see 3.6.7) shall always be performed last.

The expansions described in this clause shall occur in the same shell environment as that in which the command is executed.

If the complete expansion appropriate for a word results in an empty field, that empty field shall be deleted from the list of fields that form the completely expanded command, unless the original word contained single-quote or double-quote characters.

The `$` character is used to introduce parameter expansion, command substitution, or arithmetic evaluation. If an unquoted `$` is followed by a character that is either not numeric, the name of one of the special parameters (see 3.5.2), a valid first character of a variable name, a left curly brace (`{`), or a left parenthesis, the result is unspecified.

3.6.0.1 Word Expansions Rationale. *(This subclause is not a part of P1003.2)*

IFS is used for performing field splitting on the results of parameter and command substitution; it is not used for splitting all fields. Previous versions of the shell used it for splitting all fields during field splitting, but this has severe problems because the shell can no longer parse its own script. There are also important security implications caused by this behavior. All useful applications of **IFS** use it for parsing input of the `read` utility and for splitting the results of parameter and command substitution. New versions of the shell have fixed this bug, and POSIX.2 requires the corrected behavior.

The rule concerning expansion to a single field requires that if `foo=abc` and `bar=def`, that

```
"$foo" "$bar"
```

expands to the single field

```
abcdef
```

The rule concerning empty fields can be illustrated by:

```

595      $ unset foo
596      $ set $foo bar ' ' xyz "$foo" abc
597      $ for i
598      > do
599      >     echo "-$i-"
600      > done
601      -bar-
602      - -
603      -xyz-
604      - -
605      -abc-

```

Step (1) indicates that Tilde Expansion, Parameter Expansion, Command Substitution, and Arithmetic Expansion are all processed simultaneously as they are scanned. For example, the following is valid arithmetic:

```

609      x=1
610      echo $(( $(echo 3)+$x ))

```

An earlier draft stated that Tilde Expansion preceded the other steps, but this is not the case in known historical implementations; if it were, and a referenced home directory contained a \$ character, expansions would result within the directory name.

3.6.1 Tilde Expansion

A *tilde-prefix* consists of an unquoted tilde character at the beginning of a word, followed by all of the characters preceding the first unquoted slash in the word, or all the characters in the word if there is no slash. In an assignment (see 3.1.15), multiple tilde prefixes can be used: at the beginning of the word (i.e., following the equals-sign of the assignment) and/or following any unquoted colon. A tilde prefix in an assignment is terminated by the first unquoted colon or slash. If none of the characters in the tilde-prefix are quoted, the characters in the tilde-prefix following the tilde shall be treated as a possible login name from the user database (see POSIX.1 {8} Section 9). A portable login name cannot contain characters outside the set given in the description of the **LOGNAME** environment variable in POSIX.1 {8}. If the login name is null (i.e., the tilde-prefix contains only the tilde), the tilde-prefix shall be replaced by the value of the variable **HOME**. If **HOME** is unset, the results are unspecified. Otherwise, the tilde-prefix shall be replaced by a pathname of the home directory associated with the login name obtained using the equivalent of the POSIX.1 {8} *getpwnam()* function. If the system does not recognize the login name, the results are undefined.

3.6.1.1 Tilde Expansion Rationale. *(This subclause is not a part of P1003.2)*

The text about quoting of the word indicates that `\~hlj/`, `~h\lj/`, `~"hlj"/`, `~hlj\`, and `~hlj/` are not equivalent: only the last will cause tilde expansion.

Tilde expansion generally occurs only at the beginning of words, but POSIX.2 has adopted an exception based on historical practice in the KornShell:

```
PATH=/posix/bin:~dgg/bin
```

is eligible for tilde expansion because tilde follows a colon and none of the relevant characters is quoted. Consideration was given to prohibiting this behavior because any of the following are reasonable substitutes:

```
PATH=$(printf %s: ~rms/bin ~bfox/bin ...)
PATH=$(printf %s ~karels/bin : ~bostic/bin)
for Dir in ~maat/bin ~srb/bin ...
do
    PATH=${PATH:+$PATH:}$Dir
done
```

(In the first command, any number of directory names are concatenated and separated with colons, but it may be undesirable to end the variable with a colon because this is an obsolescent means to include dot at the end of the **PATH**. In the second, explicit colons are used for each directory. In all cases, the shell performs tilde expansion on each directory because all are separate words to the shell.)

The exception was included to avoid breaking numerous KornShell scripts and interactive users and despite the fact that variable assignments in scripts derived from other systems will have to use quoting in some cases to allow literal tildes in strings. (This latter problem should be relatively rare because only tildes preceding known login names in unquoted strings are affected.)

Note that expressions in operands such as

```
make -k mumble LIBDIR=~chet/lib
```

do not qualify as shell variable assignments and tilde expansion is not performed (unless the command does so itself, which `make` does not).

In an earlier draft, tilde expansion occurred following any unquoted equals-sign or colon, but this was removed because of its complexity and to avoid breaking commands such as:

```
rcp hostname:~marc/.profile .
```

A suggestion was made that the special sequence “`$~`” should be allowed to force tilde expansion anywhere. Since this is not historical practice, it has been left for future implementations to evaluate. (The description in 3.2 requires that a dollar-sign be quoted to represent itself, so the `$~` combination is already unspecified.)

The results of giving tilde with an unknown login name are undefined because the KornShell `~+` and `~—` constructs make use of this condition, but in general it is an error to give an incorrect login name with tilde. The results of having **HOME** unset are unspecified because some historical shells treat this as an error.

3.6.2 Parameter Expansion

The format for parameter expansion is as follows:

`${expression}`

where *expression* consists of all characters until the matching `}`. Any `}` escaped by a backslash or within a quoted string, and characters in embedded arithmetic expansions, command substitutions, and variable expansions, shall not be examined in determining the matching `}`.

The simplest form for parameter expansion is:

`${parameter}`

The value, if any, of *parameter* shall be substituted.

The parameter name or symbol can be enclosed in braces, which are optional except for positional parameters with more than one digit or when *parameter* is followed by a character that could be interpreted as part of the name. The matching closing brace shall be determined by counting brace levels, skipping over enclosed quoted strings and command substitutions.

If the parameter name or symbol is not enclosed in braces, the expansion shall use the longest valid name (see 3.1.5), whether or not the symbol represented by that name exists. If a parameter expansion occurs inside double-quotes:

- Pathname expansion shall not be performed on the results of the expansion.
- Field splitting shall not be performed on the results of the expansion, with the exception of `@`; see 3.5.2.

In addition, a parameter expansion can be modified by using one of the following formats. In each case that a value of *word* is needed (based on the state of *parameter*, as described below), *word* shall be subjected to tilde expansion, parameter expansion, command substitution, and arithmetic expansion. If *word* is not needed, it shall not be expanded. The `}` character that delimits the following parameter expansion modifications shall be determined as described previously in this subclause and in 3.2.3. (For example, `${foo-bar}xyz` would result in the expansion of `foo` followed by the string `xyz` if `foo` is set, else the string `barxyz`).

`${parameter:-word}` **Use Default Values.** If *parameter* is unset or null, the expansion of *word* shall be substituted; otherwise, the value of *parameter* shall be substituted.

709	<code>\${parameter:=word}</code>	Assign Default Values. If <i>parameter</i> is unset or null, the expansion of <i>word</i> shall be assigned to <i>parameter</i> . In all cases, the final value of <i>parameter</i> shall be substituted. Only variables, not positional parameters or special parameters, can be assigned in this way.
710		
711		
712		
713		
714		
715	<code>\${parameter:?[word]}</code>	Indicate Error if Null or Unset. If <i>parameter</i> is unset or null, the expansion of <i>word</i> (or a message indicating it is unset if <i>word</i> is omitted) shall be written to standard error and the shell shall exit with a nonzero exit status. Otherwise, the value of <i>parameter</i> shall be substituted. An interactive shell need not exit.
716		
717		
718		
719		
720		
721		
722	<code>\${parameter:+word}</code>	Use Alternate Value. If <i>parameter</i> is unset or null, null shall be substituted; otherwise, the expansion of <i>word</i> shall be substituted.
723		
724		
725	In the parameter expansions shown previously, use of the colon in the format results in a test for a parameter that is unset or null; omission of the colon results in a test for a parameter that is only unset.	
726		
727		
728	<code>\${#parameter}</code>	String Length. The length in characters of the value of <i>parameter</i> . If <i>parameter</i> is * or @, the result of the expansion is unspecified.
729		
730		
731	The following four varieties of parameter expansion provide for substring processing. In each case, pattern matching notation (see 3.13), rather than regular expression notation, shall be used to evaluate the patterns. If <i>parameter</i> is * or @, the result of the expansion is unspecified. Enclosing the full parameter expansion string in double-quotes shall not cause the following four varieties of pattern characters to be quoted, whereas quoting characters within the braces shall have this effect.	
732		
733		
734		
735		
736		
737		
738	<code>\${parameter%word}</code>	Remove Smallest Suffix Pattern. The <i>word</i> shall be expanded to produce a pattern. The parameter expansion then shall result in <i>parameter</i> , with the smallest portion of the suffix matched by the <i>pattern</i> deleted.
739		
740		
741		
742		
743	<code>\${parameter%%word}</code>	Remove Largest Suffix Pattern. The <i>word</i> shall be expanded to produce a pattern. The parameter expansion then shall result in <i>parameter</i> , with the largest portion of the suffix matched by the <i>pattern</i> deleted.
744		
745		
746		
747		
748	<code>\${parameter#word}</code>	Remove Smallest Prefix Pattern. The <i>word</i> shall be expanded to produce a pattern. The parameter expansion then shall result in <i>parameter</i> , with the smallest portion of the prefix matched by the <i>pattern</i> deleted.
749		
750		
751		
752		

753 $\${parameter}##word$ **Remove Largest Prefix Pattern.** The *word* shall
 754 be expanded to produce a pattern. The parameter
 755 expansion then shall result in *parameter*, with the
 756 largest portion of the prefix matched by the *pattern*
 757 deleted.

758 **3.6.2.1 Parameter Expansion Rationale.** (*This subclause is not a part of P1003.2*)

759 When the shell is scanning its input to determine the boundaries of a name, it is
 760 not bound by its knowledge of what names are already defined. For example, if *F*
 761 is a defined shell variable, the command "echo \$Fred" does not echo the value
 762 of \$F followed by *red*; it selects the longest possible valid name, *Fred*, which in
 763 this case might be unset.

764 The rule for finding the closing } in $\${...}$ is the one used in the KornShell and
 765 is upward compatible with the Bourne shell, which does not determine the closing
 766 } until the word is expanded. The advantage of this is that incomplete expan-
 767 sions, such as

768 $\${foo}$

769 can be determined during tokenization, rather than during expansion.

770 The four expansions with the optional colon have been hard to understand from
 771 the historical documentation. The following table summarizes the effect of the
 772 colon:

	<i><u>parameter set and not null</u></i>	<i><u>parameter set but null</u></i>	<i><u>parameter unset</u></i>	
775 $\${parameter}:-word$	substitute <i>parameter</i>	substitute <i>word</i>	substitute <i>word</i>	
777 $\${parameter}-word$	substitute <i>parameter</i>	substitute null	substitute <i>word</i>	
779 $\${parameter}:=word$	substitute <i>parameter</i>	assign <i>word</i>	assign <i>word</i>	
781 $\${parameter}=word$	substitute <i>parameter</i>	substitute <i>parameter</i>	assign <i>word</i>	
783 $\${parameter}?:word$	substitute <i>parameter</i>	error, exit	error, exit	
785 $\${parameter}?word$	substitute <i>parameter</i>	substitute null	error, exit	
787 $\${parameter}:+word$	substitute <i>word</i>	substitute null	substitute null	1
789 $\${parameter}+word$	substitute <i>word</i>	substitute <i>word</i>	substitute null	1

In all cases shown with “substitute,” the expression is replaced with the value shown. In all cases shown with “assign,” *parameter* is assigned that value, which also replaces the expression.

The string length and substring capabilities were included because of the demonstrated need for them, based on their usage in other shells, such as C-shell and KornShell.

Historical versions of the KornShell have not performed tilde expansion on the word part of parameter expansion; however, it is more consistent to do so.

Examples

```
${parameter:-word}
```

In this example, `ls` is executed only if `x` is null or unset. [The `$(ls)` command substitution notation is explained in 3.6.3.]

```
${x:-$(ls)}
```

```
${parameter:=word}
```

```
unset X
echo ${X:=abc}
abc
```

```
${parameter:?word}
```

```
unset posix
echo ${posix:?}
sh: posix: parameter null or not set
```

```
${parameter:+word}
```

```
set a b c
echo ${3:+posix}
posix
```

```
${#parameter}
```

```
HOME=/usr/posix
echo ${#HOME}
10
```

```
${parameter%word}
```

```
x=file.c
echo ${x%.c}.o
file.o
```

```
${parameter%%word}
```

```
x=posix/src/std
echo ${x%%/*}
posix
```

```

828     ${parameter#word}
829         x=$HOME/src/cmd
830         echo ${x#$HOME}
831         /src/cmd
832     ${parameter##word}
833         x=/one/two/three
834         echo ${x##*/}
835         three

```

The double-quoting of patterns is different depending on where the double-quotes are placed:

```

838     "${x#*}"          The asterisk is a pattern character.
839     "${x#"*"}"        The literal asterisk is quoted and not special.

```

3.6.3 Command Substitution

Command substitution allows the output of a command to be substituted in place of the command name itself. Command substitution shall occur when the command is enclosed as follows:

```

844     $(command)
845 or ("backquoted" version):
846     `command`

```

The shell shall expand the command substitution by executing *command* in a sub-shell environment (see 3.12) and replacing the command substitution [the text of *command* plus the enclosing `$ ()` or backquotes] with the standard output of the command, removing sequences of one or more <newline>s at the end of the substitution. (Embedded <newline>s before the end of the output shall not be removed; however, during field splitting, they may be translated into <space>s, depending on the value of **IFS** and quoting that is in effect.)

Within the backquoted style of command substitution, backslash shall retain its literal meaning, except when followed by

```

856     $ ` \
857 (dollar-sign, backquote, backslash). The search for the matching backquote shall 2
858 be satisfied by the first backquote found without a preceding backslash; during 2
859 this search, if a nonescaped backquote is encountered within a shell comment, a 2
860 here-document, an embedded command substitution of the $(command) form, or 2
861 a quoted string, undefined results occur. A single- or double-quoted string that
862 begins, but does not end, within the `...` sequence produces undefined results.

```

With the `$(command)` form, all characters following the open parenthesis to the matching closing parenthesis constitute the *command*. Any valid shell script can be used for *command*, except:

866

— A script consisting solely of redirections produces unspecified results.

2

867

— See the restriction on single subshells described below.

2

868

The results of command substitution shall not be processed for further tilde

1

869

expansion, parameter expansion, command substitution, or arithmetic expansion.

1

870

If a command substitution occurs inside double-quotes, field splitting and path-

871

name expansion shall not be performed on the results of the substitution.

872

Command substitution can be nested. To specify nesting within the backquoted

873

version, the application shall precede the inner backquotes with backslashes; for

874

example,

875

`\`command\``

876

If the command substitution consists of a single subshell, such as

877

`$((command))`

878

a conforming application shall separate the `$(` and `(` into two tokens (i.e.,

879

separate them with white space).

880

3.6.3.1 Command Substitution Rationale. *(This subclause is not a part of P1003.2)*

881

The new `$()` form of command substitution was adopted from the KornShell to

882

solve a problem of inconsistent behavior when using backquotes. For example:

883

Command	Output
<code>echo `x`</code>	<code>x</code>
<code>echo `echo `x``</code>	<code>x</code>
<code>echo \$(echo `x`)</code>	<code>x</code>

887

Additionally, the backquoted syntax has historical restrictions on the contents of

2

888

the embedded command. While the new `$()` form can process any kind of valid

2

889

embedded script, the backquoted cannot handle some valid scripts that include

2

890

backquotes. For example, these otherwise valid embedded scripts do not work in

2

891

the left column, but do work on the right:

2

892

<code>echo `</code>	<code>echo \$(</code>	2
<code>cat <<\eof</code>	<code>cat <<\eof</code>	2
<code>a here-doc with `</code>	<code>a here-doc with)</code>	2
<code>eof</code>	<code>eof</code>	2
<code>`</code>	<code>)</code>	2
897	<code>echo `</code>	2
898	<code>echo abc # a comment with `</code>	2
899	<code>`</code>	2
900	<code>echo `</code>	2
901	<code>echo ` ` `</code>	2
902	<code>`</code>	2

903

Some historical KornShell implementations did not process the first two examples

2

904

correctly, but the author has agreed to make the appropriate modifications to do

2

so. The KornShell will also be modified so that the following works:

```

    echo $(
        case word in
            [Ff]oo) echo found foo ;;
        esac
    )

```

Because of these inconsistent behaviors, the backquoted variety of command substitution is not recommended for new applications that nest command substitutions or attempt to embed complex scripts. Because of its widespread historical use, particularly by interactive users, however, the backquotes were retained in POSIX.2 without being declared obsolescent.

The KornShell feature:

If *command* is of the form *<word>*, *word* is expanded to generate a path-name, and the value of the command substitution is the contents of this file with any trailing *<newline>*s deleted.

was omitted from this standard because `$(cat word)` is an appropriate substitute. However, to prevent breaking numerous scripts relying on this feature, it is unspecified to have a script within `$()` that has only redirections.

The requirement to separate `$(` and `(` when a single subshell is command-substituted is to avoid any ambiguities with Arithmetic Expansion. See 3.6.4.1.

3.6.4 Arithmetic Expansion

Arithmetic expansion provides a mechanism for evaluating an arithmetic expression and substituting its value. The format for arithmetic expansion shall be as follows:

```
$( ( expression ) )
```

The expression shall be treated as if it were in double-quotes, except that a double-quote inside the expression is not treated specially. The shell shall expand all tokens in the expression for parameter expansion, command substitution, and quote removal.

Next, the shell shall treat this as an arithmetic expression and substitute the value of the expression. The arithmetic expression shall be processed according to the rules given in 2.9.2.1, with the following exceptions:

- (1) Only integer arithmetic is required.
- (2) The `sizeof()` operator and the prefix and postfix `++` and `--` operators are not required.
- (3) Selection, Iteration, and Jump Statements are not supported.

As an extension, the shell may recognize arithmetic expressions beyond those listed. If the expression is invalid, the expansion fails and the shell shall write a message to standard error indicating the failure.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

3.6.4.1 Arithmetic Expansion Rationale. *(This subclause is not a part of P1003.2)*

Numerous ballots were received objecting to the inclusion of the `(())` form of KornShell arithmetic in previous drafts. The developers of the standard concluded that there is a strong desire for some kind of arithmetic evaluator to replace `expr`, and that tying it in with `$` makes it fit in nicely with the standard shell language, and provides access to arithmetic evaluation in places where accessing a utility would be inconvenient or clumsy.

Following long debate by interested members of the balloting group, the syntax and semantics for arithmetic were changed. The language is essentially a pure arithmetic evaluator of constants and operators (excluding assignment) and represents a simple subset of the previous arithmetic language [which was derived from the KornShell's `(())` construct]. The syntax was changed from that of a command denoted by `((expression))`, to an expansion denoted by `$((expression))`. The new form is a dollar expansion (`$`), which evaluates the expression and substitutes the resulting value. Objections to the previous style of arithmetic included that it was too complicated, did not fit in well with the shell's use of variables, and the syntax conflicted with subshells. The justification for the new syntax is that the shell is traditionally a macro language, and if a new feature is to be added, it should be done by extending the capabilities presented by the current model of the shell, rather than by inventing a new one outside the model: adding a new dollar expansion was perceived to be the most intuitive and least destructive way to add such a new capability.

In Drafts 9 and 10, a form `$(expression)` was used. It was functionally equivalent to the `$(())` of the current text, but objections were lodged that the 1988 KornShell had already implemented `$(())` and there was no compelling reason to invent yet another syntax. Furthermore, the `$()` syntax had a minor incompatibility involving the patterns in `case` statements.

The portion of the C Standard {7} arithmetic operations selected corresponds to the operations historically supported in the KornShell.

A simple example using arithmetic expansion:

```
# repeat a command 100 times
x=100
while [ $x -gt 0 ]
do
    command
    x=$((x-1))
done
```

It was concluded that the `test` command (`[]`) was sufficient for the majority of relational arithmetic tests, and that tests involving complicated relational expressions within the shell are rare, yet could still be accommodated by testing the value of `$(())` itself. For example:

```
# a complicated relational expression
while [ $(( ($x + $y)/($a * $b)) < ($foo*$bar) )) -ne 0 ]
```

or better yet, the rare script that has many complex relational expressions could

define a function like this:

```
val() {
    return $(!$1)
}
```

and complicated tests would be less intimidating:

```
while val $(( ($x + $y)/($a * $b) < ($foo*$bar) ))
do
    # some calculations
done
```

Another suggestion was to modify `true` and `false` to take an optional argument, and `true` would exit `true` only if the argument is nonzero, and `false` would exit `false` only if the argument is nonzero. The suggestion was not favorably received by the balloting group (those contacted were negative about it, all others were silent in their latest ballots).

```
while true $(( $x > 5 && $y <= 25 ))
```

There is a minor portability concern with the new syntax. The example `$((2+2))` could have been intended to mean a command substitution of a utility named `2+2` in a subshell. The developers of POSIX.2 consider this to be obscure and isolated to some KornShell scripts [because `$()` command substitution existed previously only in the KornShell]. The text on Command Substitution has been changed to require that the `$()` and `(` be separate tokens if this usage is needed.

An example such as

```
echo $((echo hi);(echo there))
```

should not be misinterpreted by the shell as arithmetic because attempts to balance the parentheses pairs would indicate that they are subshells. However, as indicated by 3.1.1, a conforming application must separate two adjacent parentheses with white space to indicate nested subshells.

3.6.5 Field Splitting

After parameter expansion (3.6.2), command substitution (3.6.3), and arithmetic expansion (3.6.4) the shell shall scan the results of expansions and substitutions that did not occur in double-quotes for field splitting and multiple fields can result.

The shell shall treat each character of the **IFS** as a delimiter and use the delimiters to split the results of parameter expansion and command substitution into fields.

- (1) If the value of **IFS** is `<space>`, `<tab>`, and `<newline>`, or if it is unset, any sequence of `<space>`, `<tab>`, or `<newline>` characters at the beginning or end of the input shall be ignored and any sequence of those characters within the input shall delimit a field. (For example, the input

- 1028 <newline><space><tab>foo<tab><tab>bar<space>
 1029 yields two fields, foo and bar).
- 1030 (2) If the value of **IFS** is null, no field splitting shall be performed.
- 1031 (3) Otherwise, the following rules shall be applied in sequence. The term 1
 1032 “**IFS** white space” is used to mean any sequence (zero or more instances) 1
 1033 of white-space characters that are in the **IFS** value (e.g., if **IFS** contains 1
 1034 <space><comma><tab>, any sequence of <space> and <tab> charac- 1
 1035 ters is considered **IFS** white space). 1
- 1036 (a) **IFS** white space shall be ignored at the beginning and end of the 1
 1037 input. 1
- 1038 (b) Each occurrence in the input of an **IFS** character that is not **IFS** 1
 1039 white space, along with any adjacent **IFS** white space, shall delimit 1
 1040 a field, as described previously. 1
- 1041 (c) Nonzero-length **IFS** white space shall delimit a field. 1

1042 **3.6.5.1 Field Splitting Rationale.** *(This subclause is not a part of P1003.2)*

1043 The operation of field splitting using **IFS** as described in earlier drafts was based
 1044 on the way the KornShell splits words, but is incompatible with other common
 1045 versions of the shell. However, each has merit, and so a decision was made to
 1046 allow both. If the **IFS** variable is unset, or is <space><tab><newline>, the
 1047 operation is equivalent to the way the System V shell splits words. Using charac-
 1048 ters outside the <space><tab><newline> set yields the KornShell behavior,
 1049 where each of the non-<space><tab><newline> characters is significant. This
 1050 behavior, which affords the most flexibility, was taken from the way the original
 1051 awk handled field splitting.

1052 The (3) rule can be summarized as a pseudo ERE: 1

1053 (s*ns*|s+) 1

1054 where s is an **IFS** white-space character and n is a character in the **IFS** that is not 1
 1055 white space. Any string matching that ERE delimits a field, except that the s+ 1
 1056 form does not delimit fields at the beginning or the end of a line. For example, if 1
 1057 **IFS** is <space><comma>, the string 1

1058 <space><space>red<space><space>, <space>white<space>blue 1

1059 yields the three colors as the delimited fields. 1

1060 **3.6.6 Pathname Expansion**

1061 After field splitting, if set -f is not in effect, each field in the resulting command
 1062 line shall be expanded using the algorithm described in 3.13, qualified by the
 1063 rules in 3.13.3.

3.6.7 Quote Removal

The quote characters

\ ' "

(backslash, single-quote, double-quote) that were present in the original word shall be removed unless they have themselves been quoted.

3.7 Redirection

Redirection is used to open and close files for the current shell execution environment (see 3.12) or for any command. *Redirection operators* can be used with numbers representing file descriptors (see the definition in POSIX.1 {8}) as described below. See also 2.9.1. The relationship between these file descriptors and access to them in a programming language is specified in the language binding for that language to this standard.

The overall format used for redirection is:

`[n]redir-op word`

The number *n* is an optional decimal number designating the file descriptor number; it shall be delimited from any preceding text and immediately precede the redirection operator *redir-op*. If *n* is quoted, the number shall not be recognized as part of the redirection expression. (For example, `echo \2>a` writes the character 2 into file a). If any part of *redir-op* is quoted, no redirection expression shall be recognized. (For example, `echo 2\>a` writes the characters 2>a to standard output.) The optional number, redirection operator, and *word* shall not appear in the arguments provided to the command to be executed (if any).

In this standard, open files are represented by decimal numbers starting with zero. It is implementation defined what the largest value can be; however, all implementations shall support at least 0 through 9 for use by the application. These numbers are called *file descriptors*. The values 0, 1, and 2 have special meaning and conventional uses and are implied by certain redirection operations; they are referred to as *standard input*, *standard output*, and *standard error*, respectively. Programs usually take their input from standard input, and write output on standard output. Error messages are usually written to standard error. The redirection operators can be preceded by one or more digits (with no intervening <blank>s allowed) to designate the file descriptor number.

If the redirection operator is << or <<-, the word that follows the redirection operator shall be subjected to quote removal; it is unspecified whether any of the other expansions occur. For the other redirection operators, the word that follows the redirection operator shall be subjected to tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote removal. Pathname expansion shall not be performed on the word by a noninteractive shell; an interactive shell may perform it, but shall do so only when the expansion would result in one word.

1104 If more than one redirection operator is specified with a command, the order of
1105 evaluation is from beginning to end.

1106 In the following description of redirections, references are made to opening and
1107 creating files. These references shall conform to the requirements in 2.9.1.4. A
1108 failure to open or create a file shall cause the redirection to fail.

1109 3.7.1 Redirecting Input

1110 Input redirection shall cause the file whose name results from the expansion of
1111 *word* to be opened for reading on the designated file descriptor, or standard input
1112 if the file descriptor is not specified.

1113 The general format for redirecting input is:

1114 `[n]<word`

1115 where the optional *n* represents the file descriptor number. If the number is omit-
1116 ted, the redirection shall refer to standard input (file descriptor 0).

1117 3.7.2 Redirecting Output

1118 The two general formats for redirecting output are:

1119 `[n]>word`

1120 `[n]>|word`

1121 where the optional *n* represents the file descriptor number. If the number is omit-
1122 ted, the redirection shall refer to standard output (file descriptor 1).

1123 Output redirection using the > format shall fail if the *noclobber* option is set (see 1
1124 the description of `set -C` in 3.14.11) and the file named by the expansion of *word* 1
1125 exists and is a regular file. Otherwise, redirection using the > or >| formats shall 1
1126 cause the file whose name results from the expansion of *word* to be created and
1127 opened for output on the designated file descriptor, or standard output if none is
1128 specified. If the file does not exist, it shall be created; otherwise, it shall be trun-
1129 cated to be an empty file after being opened.

1130 3.7.3 Appending Redirected Output

1131 Appended output redirection shall cause the file whose name results from the
1132 expansion of *word* to be opened for output on the designated file descriptor. The
1133 file is opened as if the POSIX.1 {8} *open()* function was called with the O_APPEND
1134 flag. If the file does not exist, it shall be created.

1135 The general format for appending redirected output is as follows:

1136 `[n]>>word`

1137 where the optional *n* represents the file descriptor number.

3.7.4 Here-Document

The redirection operators `<<` and `<<-` both allow redirection of lines contained in a shell input file, known as a *here-document*, to the standard input of a command.

The here-document shall be treated as a single word that begins after the next `<newline>` and continues until there is a line containing only the delimiter, with no trailing `<blank>`s. Then the next here-document starts, if there is one. The format is as follows:

```
[n]<<word
      here-document
      delimiter
```

If any character in *word* is quoted, the delimiter shall be formed by performing quote removal on *word*, and the here-document lines shall not be expanded. Otherwise, the delimiter shall be the *word* itself.

If no characters in *word* are quoted, all lines of the here-document shall be expanded for parameter expansion, command substitution, and arithmetic expansion. In this case, the backslash in the input shall behave as the backslash inside double-quotes (see 3.2.3). However, the double-quote character (") shall not be treated specially within a here-document, except when the double-quote appears within `$ ()`, `` ``, or `${ }`.

1

If the redirection symbol is `<<-`, all leading `<tab>` characters shall be stripped from input lines and the line containing the trailing delimiter. If more than one `<<` or `<<-` operator is specified on a line, the here-document associated with the first operator shall be supplied first by the application and shall be read first by the shell.

3.7.5 Duplicating an Input File Descriptor

The redirection operator

```
[n]<&word
```

is used to duplicate one input file descriptor from another, or to close one. If *word* evaluates to one or more digits, the file descriptor denoted by *n*, or standard input if *n* is not specified, shall be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent a file descriptor already open for input, a redirection error shall result (see 3.8.1). If *word* evaluates to `-`, file descriptor *n*, or standard input if *n* is not specified, shall be closed. If *word* evaluates to something else, the behavior is unspecified.

1

3.7.6 Duplicating an Output File Descriptor

The redirection operator

`[n]>&word`

is used to duplicate one output file descriptor from another, or to close one. If *word* evaluates to one or more digits, the file descriptor denoted by *n*, or standard output if *n* is not specified, shall be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent a file descriptor already open for output, a redirection error shall result (see 3.8.1). If *word* evaluates to `–`, file descriptor *n*, or standard output if *n* is not specified, shall be closed. If *word* evaluates to something else, the behavior is unspecified.

3.7.7 Open File Descriptors for Reading and Writing.

The redirection operator

`[n]<>word`

shall cause the file whose name is the expansion of *word* to be opened for both reading and writing on the file descriptor denoted by *n*, or standard input if *n* is not specified. If the file does not exist, it shall be created.

3.7.8 Redirection Rationale. *(This subclause is not a part of P1003.2)*

In the C binding for POSIX.1 {8}, file descriptors are integers in the range 0 – ({OPEN_MAX}–1). The file descriptors discussed in Redirection are that same set of small integers.

As POSIX.2 is being finalized, it is not known how file descriptors will be represented in the language-independent description of POSIX.1 {8}. The current consensus appears to be that they will remain as small integers, but it is still possible that they will be defined as an opaque type. If they remain as integers, then the current POSIX.2 wording is acceptable. If they become an opaque type, then the C binding to POSIX.1 {8} will have to define the mapping from the binding's small integers to the opaque type, and the Redirection clause in POSIX.2 will have to be modified to specify that same mapping.

Having multidigit file descriptor numbers for I/O redirection can cause some obscure compatibility problems. Specifically, scripts that depend on an example command:

```
echo 22>/dev/null
```

echoing "2" are somewhat broken to begin with. However, the file descriptor number still must be delimited from the preceding text. For example,

```
cat file2>foo
```

will write the contents of `file2`, not the contents of `file`.

The `>|` format of output redirection was adopted from the KornShell. Along with the *noclobber* option, `set -C`, it provides a safety feature to prevent inadvertent overwriting of existing files. (See the rationale with the `pathchk` utility for why this step was taken.) The restriction on regular files is historical practice.

The System V shell and the KornShell have differed historically on pathname expansion of *word*; the former never performed it, the latter only when the result was a single field (file). As a compromise, it was decided that the KornShell functionality was useful, but only as a shorthand device for interactive users. No reasonable shell script would be written with a command such as:

```
cat foo > a*
```

Thus, shell scripts are prohibited from doing it, while interactive users can select the shell with which they are most comfortable.

The construct `2>&1` is often used to redirect standard error to the same file as standard output. Since the redirections take place beginning to end, the order of redirections is significant. For example:

```
ls > foo 2>&1
```

directs both standard output and standard error to file `foo`. However

```
ls 2>&1 > foo
```

only directs standard output to file `foo` because standard error was duplicated as standard output before standard output was directed to file `foo`.

The `<>` operator is a feature first documented in the KornShell, but it has been silently present in both System V and BSD shells. It could be useful in writing an application that worked with several terminals, and occasionally wanted to start up a shell. That shell would in turn be unable to run applications that run from an ordinary controlling terminal unless it could make use of `<>` redirection. The specific example is a historical version of the pager `more`, which reads from standard error to get its commands, so standard input and standard output are both available for their usual usage. There is no way of saying the following in the shell without `<>`:

```
cat food | more - >/dev/tty03 2<>/dev/tty03
```

Another example of `<>` is one that opens `/dev/tty` on file descriptor 3 for reading and writing:

```
exec 3<> /dev/tty
```

An example of creating a lock file for a critical code region:

```
set -C
until 2> /dev/null > lockfile
do    sleep 30
done
set +C
perform critical function
rm lockfile
```

1249 Since `/dev/null` is not a regular file, no error is generated by redirecting to it in
1250 *noclobber* mode.

1251 The case of a missing delimiter at the end of a here-document is not specified.
1252 This is considered an error in the script (one that sometimes can be difficult to
1253 diagnose), although some systems have treated end-of-file as an implicit delimiter.

1254 Tilde expansion is not performed on a here-document because the data is treated
1255 as if it were enclosed in double-quotes.

1
1

1256 3.8 Exit Status and Errors

1257 3.8.1 Consequences of Shell Errors

1258 For a noninteractive shell, an error condition encountered by a special built-in
1259 (see 3.14) or other type of utility shall cause the shell to write a diagnostic mes-
1260 sage to standard error and exit as shown in the following table:

	<u>Special Built-in</u>	<u>Other Utilities</u>
1261		
1262 Shell language syntax error	shall exit	shall exit
1263 Utility syntax error (option or operand	shall exit	shall not exit
1264 error)		
1265 Redirection error	shall exit	shall not exit
1266 Variable assignment error	shall exit	shall not exit
1267 Expansion error	shall exit	shall exit
1268 Command not found	n/a	may exit
1269 <code>dot</code> script not found	shall exit	n/a

1270 An “expansion error” is one that occurs when the shell expansions defined in 3.6
1271 are carried out (e.g., `${x!y}`, because `!` is not a valid operator); an implementa-
1272 tion may treat these as syntax errors if it is able to detect them during tokeniza-
1273 tion, rather than during expansion.

1274 If any of the errors shown as “shall (may) exit” occur in a subshell, the subshell
1275 shall (may) exit with a nonzero status, but the script containing the subshell shall
1276 not exit because of the error.

1277 In all of the cases shown in the table, an interactive shell shall write a diagnostic
1278 message to standard error without exiting.

1279 3.8.2 Exit Status for Commands

1280 Each command has an exit status that can influence the behavior of other shell
1281 commands. The exit status of commands that are not utilities are documented in
1282 this subclause. The exit status of the standard utilities are documented in their
1283 respective clauses.

If a command is not found by the shell, the exit status shall be 127. If the command name is found, but it is not an executable utility, the exit status shall be 126. See 3.9.1.1. Applications that invoke utilities without using the shell should use these exit status values to report similar errors.

If a command fails during word expansion or redirection, its exit status shall be greater than zero.

Internally, for purposes of deciding if a command exits with a nonzero exit status, the shell shall recognize the entire status value retrieved for the command by the equivalent of the POSIX.1 `{8} wait()` function `WEXITSTATUS` macro. When reporting the exit status with the special parameter `?`, the shell shall report the full eight bits of exit status available. The exit status of a command that terminated because it received a signal shall be reported as greater than 128.

3.8.3 Exit Status and Errors Rationale. *(This subclause is not a part of P1003.2)*

There is a historical difference in `sh` and `ksh` noninteractive error behavior. When a command named in a script is not found, some implementations of `sh` exit immediately, but `ksh` continues with the next command. Thus, POSIX.2 says that the shell “may” exit in this case. This puts a small burden on the programmer, who will have to test for successful completion following a command if it is important that the next command not be executed if the previous was not found. If it is important for the command to have been found, it was probably also important for it to complete successfully. The test for successful completion would not need to change.

Historically, shells have returned an exit status of $128+n$, where n represents the signal number. Since signal numbers are not standardized, there is no portable way to determine which signal caused the termination. Also, it is possible for a command to exit with a status in the same range of numbers that the shell would use to report that the command was terminated by a signal. Implementations are encouraged to chose exit values greater than 256 to indicate programs that terminated by a signal so that the exit status cannot be confused with an exit status generated by a normal termination.

Historical shells make the distinction between “utility not found” and “utility found but cannot execute” in their error messages. By specifying two seldomly used exit status values for these cases, 127 and 126 respectively, this gives an application the opportunity to make use of this distinction without having to parse an error message that would probably change from locale to locale. The POSIX.2 command, `env`, `nohup`, and `xargs` utilities also have been specified to use this convention.

When a command fails during word expansion or redirection, most historical implementations exit with a status of 1. However, there was some sentiment that this value should probably be much higher, so that an application could distinguish this case from the more normal exit status values. Thus, the language “greater than zero” was selected to allow either method to be implemented.

3.9 Shell Commands

This clause describes the basic structure of shell commands. The following command descriptions each describe a format of the command that is only used to aid the reader in recognizing the command type, and does not formally represent the syntax. Each description discusses the semantics of the command; for a formal description of the command language, consult the grammar in 3.10.

A *command* is one of the following:

- *simple command* (see 3.9.1)
- *pipeline* (see 3.9.2)
- *list* or *compound-list* (see 3.9.3)
- *compound command* (see 3.9.4)
- *function definition* (see 3.9.5).

Unless otherwise stated, the exit status of a command is that of the last simple command executed by the command. There is no limit on the size of any shell command other than that imposed by the underlying system (memory constraints, {ARG_MAX}, etc.).

3.9.0.1 Shell Commands Rationale. *(This subclause is not a part of P1003.2)*

A description of an “empty command” was removed from an earlier draft because it is only relevant in the cases of `sh -c ""`, `system("")`, or an empty shell-script file (such as the implementation of `true` on some historical systems). Since it is no longer mentioned in POSIX.2, it falls into the silently unspecified category of behavior where implementations can continue to operate as they have historically, but conforming applications will not construct empty commands. (However, note that `sh` does explicitly state an exit status for an empty string or file.) In an interactive session or a script with other commands, extra `<newline>s` or semicolons, such as

```
$ false
$
$ echo $?
1
```

would not qualify as the empty command described here because they would be consumed by other parts of the grammar.

3.9.1 Simple Commands

A *simple command* is a sequence of optional variable assignments and redirections, in any sequence, optionally followed by words and redirections, terminated by a control operator.

When a given simple command is required to be executed (i.e., when any conditional construct such as an AND-OR list or a `case` statement has not bypassed the

simple command), the following expansions, assignments, and redirections shall all be performed from the beginning of the command text to the end. 1

(1) The words that are recognized as variable assignments or redirections according to 3.10.2 are saved for processing in steps (3) and (4).

(2) The words that are not variable assignments or redirections shall be expanded. If any fields remain following their expansion, the first field shall be considered the command name, and remaining fields shall be the arguments for the command.

(3) Redirections shall be performed as described in 3.7.

(4) Each variable assignment shall be expanded for tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote removal prior to assigning the value.

In the preceding list, the order of steps (3) and (4) may be reversed for the processing of special built-in utilities. See 3.14.

If no command name results, variable assignments shall affect the current execution environment. Otherwise, the variable assignments shall be exported for the execution environment of the command and shall not affect the current execution environment (except for special built-ins). If any of the variable assignments attempt to assign a value to a read-only variable, a variable assignment error shall occur. See 3.8.1 for the consequences of these errors.

If there is no command name, any redirections shall be performed in a subshell environment; it is unspecified whether this subshell environment is the same one as that used for a command substitution within the command. [To affect the current execution environment, see `exec` (3.14.6)]. If any of the redirections performed in the current shell execution environment fail, the command shall immediately fail with an exit status greater than zero, and the shell shall write an error message indicating the failure. See 3.8.1 for the consequences of these failures on interactive and noninteractive shells.

If there is a command name, execution shall continue as described in 3.9.1.1. If there is no command name, but the command contained a command substitution, the command shall complete with the exit status of the last command substitution performed. Otherwise, the command shall complete with a zero exit status.

3.9.1.0.1 Simple Commands Rationale. *(This subclause is not a part of P1003.2)*

The enumerated list is used only when the command is actually going to be executed. For example, in: 1

```
true || $foo *
```

no expansions are performed. 1

The following example illustrates both how a variable assignment without a command name affects the current execution environment, and how an assignment with a command name only affects the execution environment of the command.

```

1404      $ x=red
1405      $ echo $x
1406      red
1407      $ export x
1408      $ sh -c 'echo $x'
1409      red
1410      $ x=blue sh -c 'echo $x'
1411      blue
1412      $ echo $x
1413      red

```

1414 This next example illustrates that redirections without a command name are still
 1415 performed.

```

1416      $ ls foo
1417      ls: foo: no such file or directory
1418      $ > foo
1419      $ ls foo
1420      foo

```

1421 Historical practice is for a command without a command name, but that includes
 1422 a command substitution, to have an exit status of the last command substitution
 1423 that the shell performed and some historical scripts rely on this. For example:

```

1424      if      x=$(command)
1425      then    ...
1426      fi

```

1427 An example of redirections without a command name being performed in a sub-
 1428 shell shows that the here-document does not disrupt the standard input of the
 1429 while loop:

```

1430      IFS=:
1431      while  read a b
1432      do     echo $a
1433            <<-eof
1434            Hello
1435            eof
1436      done </etc/passwd

```

1437 Some examples of commands without command names in AND/OR lists:

```

1438      > foo || {
1439          echo "error: foo cannot be created" >&2
1440          exit 1
1441      }

```

```

1442      # set saved if /vmunix.save exists
1443      test -f /vmunix.save && saved=1

```

1444 Command substitution and redirections without command names both occur in
 1445 subshells, but they are not the same ones. For example, in: 1

```

1446      exec 3> file
1447      var=$(echo foo >&3) 3>&1

```

1448 it is unspecified whether `foo` will be echoed to the file or to standard output.

1449 3.9.1.1 Command Search and Execution

1450 If a simple command results in a command name and an optional list of argu-
1451 ments, the following actions shall be performed.

1452 (1) If the command name does not contain any slashes, the first successful
1453 step in the following sequence shall occur:

1454 (a) If the command name matches the name of a special built-in utility,
1455 that special built-in utility shall be invoked.

1456 (b) If the command name matches the name of a function known to this
1457 shell, the function shall be invoked as described in 3.9.5. [If the
1458 implementation has provided a standard utility in the form of a
1459 function, it shall not be recognized at this point. It shall be invoked
1460 in conjunction with the path search in step (1)(d).]

1461 (c) If the command name matches the name of a utility listed in
1462 Table 2-2 (see 2.3), that utility shall be invoked.

1463 (d) Otherwise, the command shall be searched for using the **PATH**
1464 environment variable as described in 2.6:

1465 [1] If the search is successful:

1466 [a] If the system has implemented the utility as a regular
1467 built-in or as a shell function, it shall be invoked at this
1468 point in the path search.

1469 [b] Otherwise, the shell shall execute the utility in a 1
1470 separate utility environment (see 3.12) with actions 1
1471 equivalent to calling the POSIX.1 {8} *execve()* function
1472 with the *path* argument set to the pathname resulting
1473 from the search, *arg0* set to the command name, and the
1474 remaining arguments set to the operands, if any.

1475 If the *execve()* function fails due to an error equivalent to
1476 the POSIX.1 {8} error [ENOEXEC], the shell shall execute
1477 a command equivalent to having a shell invoked with the
1478 command name as its first operand, along with any
1479 remaining arguments passed along. If the executable file
1480 is not a text file, the shell may bypass this command exe-
1481 cution, write an error message, and return an exit status
1482 of 126. 1

1483 Once a utility has been searched for and found (either as a
1484 result of this specific search or as part of an unspecified shell
1485 startup activity), an implementation may remember its loca-
1486 tion and need not search for the utility again unless the **PATH**
1487 variable has been the subject of an assignment. If the remem-
1488 bered location fails for a subsequent invocation, the shell shall

1489 repeat the search to find the new location for the utility, if
1490 any.

1491 [2] If the search is unsuccessful, the command shall fail with an
1492 exit status of 127 and the shell shall write an error message.

1493 (2) If the command name does contain slashes, the shell shall execute the
1494 utility in a separate utility environment with actions equivalent to cal- 1
1495 ling the POSIX.1 {8} *execve()* function with the *path* and *arg0* arguments
1496 set to the command name, and the remaining arguments set to the
1497 operands, if any.

1498 If the *execve()* function fails due to an error equivalent to the POSIX.1 {8}
1499 error [ENOEXEC], the shell shall execute a command equivalent to hav-
1500 ing a shell invoked with the command name as its first operand, along
1501 with any remaining arguments passed along. If the executable file is not
1502 a text file, the shell may bypass this command execution, write an error
1503 message, and return an exit status of 126. 1

1504 **3.9.1.1.1 Command Search and Execution Rationale.** (*This subclause is not a part* 1505 *of P1003.2*)

1506 This description requires that the shell can execute shell scripts directly, even if
1507 the underlying system does not support the common `#!` interpreter convention.
1508 That is, if file `foo` contains shell commands and is executable, the following will
1509 execute `foo`:

1510 `./foo`

1511 The command search shown here does not match all historical implementations.
1512 A more typical sequence has been:

1513 — Any built-in, special or regular.

1514 — Functions.

1515 — Path search for executable files.

1516 But there are problems with this sequence. Since the programmer has no idea in
1517 advance which utilities might have been built into the shell, a function cannot be
1518 used to portably override a utility of the same name. (For example, a function
1519 named `cd` cannot be written for many historical systems.) Furthermore, the
1520 **PATH** variable is partially ineffective in this case and only a pathname with a
1521 slash can be used to ensure a specific executable file is invoked.

1522 The sequence selected for POSIX.2 acknowledges that special built-ins cannot be
1523 overridden, but gives the programmer full control over which versions of other
1524 utilities are executed. It provides a means of suppressing function lookup (via the
1525 `command` utility; see 4.12) for the user's own functions and ensures that any regu-
1526 lar built-ins or functions provided by the implementation are under the control of
1527 the path search. The mechanisms for associating built-ins or functions with exe-
1528 cutable files in the path are not specified by POSIX.2, but the wording requires
1529 that if either is implemented, the application will not be able to distinguish a
1530 function or built-in from an executable (other than in terms of performance,

presumably). The implementation must ensure that all effects specified by POSIX.2 resulting from the invocation of the regular built-in or function (interaction with the environment, variables, traps, etc.) are identical to those resulting from the invocation of an executable file.

Example: Consider three versions of the `ls` utility:

- The application includes a shell function named `ls`.
- The user writes her own utility named `ls` and puts it in `/hsa/bin`.
- The example implementation provides `ls` as a regular shell built-in that will be invoked (either by the shell or directly by *exec*) when the path search reaches the directory `/posix/bin`.

If `PATH=/posix/bin`, various invocations yield different versions of `ls`:

Invocation	Version of <code>ls</code>
<code>ls</code> (from within application script)	(1) function
command <code>ls</code> (from within application script)	(3) built-in
<code>ls</code> (from within makefile called by application)	(3) built-in
<code>system("ls")</code>	(3) built-in
<code>PATH="/hsa/bin:\$PATH" ls</code>	(2) user's version

After the *execve()* failure described, the shell normally executes the file as a shell script. Some implementations, however, attempt to detect whether the file is actually a script and not an executable from some other architecture. The method used by the KornShell is allowed by the text that indicates nontext files may be bypassed.

3.9.2 Pipelines

A *pipeline* is a sequence of one or more commands separated by the control operator `|`. The standard output of all but the last command shall be connected to the standard input of the next command.

The format for a pipeline is:

`[!] command1 [| command2 ...]`

The standard output of *command1* shall be connected to the standard input of *command2*. The standard input, standard output, or both of a command shall be considered to be assigned by the pipeline before any redirection specified by redirection operators that are part of the command (see 3.7).

If the pipeline is not in the background (see 3.9.3.1), the shell shall wait for the last command specified in the pipeline to complete, and may also wait for all commands to complete.

Exit Status

If the reserved word `!` does not precede the pipeline, the exit status shall be the exit status of the last command specified in the pipeline. Otherwise, the exit status is the logical NOT of the exit status of the last command. That is, if the last command returns zero, the exit status shall be 1; if the last command returns greater than zero, the exit status is zero.

3.9.2.1 Pipelines Rationale. *(This subclause is not a part of P1003.2)*

Because pipeline assignment of standard input or standard output or both takes place before redirection, it can be modified by redirection. For example:

```
$ command1 2>&1 | command2
```

sends both the standard output and standard error of `command1` to the standard input of `command2`.

The reserved word `!` was added to allow more flexible testing using AND and OR lists.

It was suggested that it would be better to return a nonzero value if any command in the pipeline terminates with nonzero status (perhaps the bitwise OR of all return values). However, the choice of the last-specified command semantics are historical practice and would cause application breakage if changed. An example of historical (and POSIX.2) behavior:

```
$ sleep 5 | (exit 4)
$ echo $?
4
$ (exit 4) | sleep 5
$ echo $?
0
```

1
1
1

3.9.3 Lists

An *AND-OR-list* is a sequence of one or more pipelines separated by the operators

```
&&    ||
```

A *list* is a sequence of one or more AND-OR-lists separated by the operators

```
;      &
```

and optionally terminated by

```
;      &      <newline>
```

The operators `&&` and `||` shall have equal precedence and shall be evaluated from beginning to end.

A `;` or `<newline>` terminator shall cause the preceding AND-OR-list to be executed sequentially; an `&` shall cause asynchronous execution of the preceding AND-OR-list.

1603 The term *compound-list* is derived from the grammar in 3.10; it is equivalent to a
 1604 sequence of *lists*, separated by <newline>s, that can be preceded or followed by
 1605 an arbitrary number of <newline>s.

1606 **3.9.3.0.1 Lists Rationale.** *(This subclause is not a part of P1003.2)*

1607 The equal precedence of && and || is historical practice. The developers of the
 1608 standard evaluated the model used more frequently in high level programming
 1609 languages, such as C, to allow the shell logical operators to be used for complex
 1610 expressions in an unambiguous way, but could not in the end allow existing
 1611 scripts to break in the subtle way unequal precedence might cause. Some argu-
 1612 ments were posed concerning the { } or () groupings that are required histori-
 1613 cally. There are some disadvantages to these groupings:

- 1614 — The () can be expensive, as they spawn other processes on some systems.
 1615 This performance concern is primarily an implementation issue.
- 1616 — The { } braces are not operators (they are reserved words) and require a
 1617 trailing space after each {, and a semicolon before each }. Most program-
 1618 mers (and certainly interactive users) have avoided braces as grouping con-
 1619 structs because of the irritating syntax required. Braces were not changed
 1620 to operators because that would generate compatibility issues even greater
 1621 than the precedence question; braces appear outside the context of a key-
 1622 word in many shell scripts.

1623 An example reiterates the precedence of the lists as they associate from beginning 1
 1624 to end. Both of the following commands write solely bar to standard output: 1

```
1625     false && echo foo || echo bar 1
1626     true || echo foo && echo bar 1
```

1627 The following is an example that illustrates <newline>s in compound-lists:

```
1628     while
1629         # a couple of newlines

1630         # a list
1631         date && who || ls; cat file
1632         # a couple of newlines

1633         # another list
1634         wc file > output & true

1635     do
1636         # 2 lists
1637         ls
1638         cat file
1639     done
```


3.9.3.1 Asynchronous Lists

If a command is terminated by the control operator ampersand (&), the shell shall execute the command asynchronously in a subshell. This means that the shell shall not wait for the command to finish before executing the next command.

The format for running a command in background is:

```
command1 & [command2 & ...]
```

The standard input for an asynchronous list, before any explicit redirections are performed, shall be considered to be assigned to a file that has the same properties as /dev/null. If it is an interactive shell, this need not happen. In all cases, explicit redirection of standard input shall override this activity.

When an element of an asynchronous list (the portion of the list ended by an ampersand, such as *command1*, above) is started by the shell, the process ID of the last command in the asynchronous list element shall become known in the current shell execution environment; see 3.12. This process ID shall remain known until:

- The command terminates and the application waits for the process ID, or
- Another asynchronous list is invoked before \$! (corresponding to the previous asynchronous list) is expanded in the current execution environment.

The implementation need not retain more than the {CHILD_MAX} most recent entries in its list of known process IDs in the current shell execution environment.

Exit Status

The exit status of an asynchronous list shall be zero.

3.9.3.1.1 Asynchronous Lists Rationale. *(This subclause is not a part of P1003.2)*

The grammar treats a construct such as

```
foo & bar & bam &
```

as one “asynchronous list,” but since the status of each element is tracked by the shell, the term “element of an asynchronous list” was introduced to identify just one of the *foo*, *bar*, *bam* portions of the overall list.

Unless the implementation has an internal limit, such as {CHILD_MAX}, on the retained process IDs, it would require unbounded memory for the following example:

```
while true
do    foo & echo $!
done
```

The treatment of the signals SIGINT and SIGQUIT with asynchronous lists is described in 3.11.

Since the connection of the input to the equivalent of /dev/null is considered to occur before redirections, the following script would produce no output:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
1678         exec < /etc/passwd
1679         cat <&0 &
1680         wait
```

1681 3.9.3.2 Sequential Lists

1682 Commands that are separated by a semicolon (;) shall be executed sequentially.

1683 The format for executing commands sequentially is:

1684 *command1* [*;* *command2*] ...

1685 Each command shall be expanded and executed in the order specified.

1686 Exit Status

1687 The exit status of a sequential list shall be the exit status of the last command in
1688 the list.

1689 3.9.3.3 AND Lists

1690 The control operator && shall denote an AND list. The format is:

1691 *command1* [&& *command2*] ...

1692 First *command1* is executed. If its exit status is zero, *command2* is executed, and
1693 so on until a command has a nonzero exit status or there are no more commands
1694 left to execute. The commands shall be expanded only if they are executed.

1695 Exit Status

1696 The exit status of an AND list shall be the exit status of the last command that is
1697 executed in the list.

1698 3.9.3.4 OR Lists

1699 The control operator || shall denote an OR List. The format is:

1700 *command1* [|| *command2*] ...

1701 First, *command1* is executed. If its exit status is nonzero, *command2* is executed,
1702 and so on until a command has a zero exit status or there are no more commands
1703 left to execute.

1704 Exit Status

1705 The exit status of an OR list shall be the exit status of the last command that is
1706 executed in the list.

3.9.4 Compound Commands

The shell has several programming constructs that are *compound commands*, which provide control flow for commands. Each of these compound commands has a reserved word or control operator at the beginning, and a corresponding terminator reserved word or operator at the end. In addition, each can be followed by redirections on the same line as the terminator. Each redirection shall apply to all the commands within the compound command that do not explicitly override that redirection.

3.9.4.1 Grouping Commands

The format for grouping commands is as follows:

(*compound-list*) Execute *compound-list* in a subshell environment; see 3.12. Variable assignments and built-in commands that affect the environment shall not remain in effect after the list finishes.

{ *compound-list*; } Execute *compound-list* in the current process environment.

Exit Status

The exit status of a grouping command shall be the exit status of *list*.

3.9.4.1.1 Grouping Commands Rationale. (*This subclause is not a part of P1003.2*)

The semicolon shown in { *compound-list*; } is an example of a control operator delimiting the } reserved word. Other delimiters are possible, as shown in 3.10; <newline> is frequently used.

A proposal was made to use the <do-done> construct in all cases where command grouping performed in the current process environment is performed, identifying it as a construct for the grouping commands, as well as for shell functions. This was not included because the shell already has a grouping construct for this purpose ({ }), and changing it would have been counter-productive.

3.9.4.2 for Loop

The `for` loop shall execute a sequence of commands for each member in a list of *items*. The `for` loop requires that the *reserved words* `do` and `done` be used to delimit the sequence of commands.

The format for the `for` loop is as follows.

```
for name [ in word ... ]
do
    compound-list
done
```

1743 First, the list of words following `in` shall be expanded to generate a list of items.
 1744 Then, the variable *name* shall be set to each item, in turn, and the *compound-list*
 1745 executed each time. If no items result from the expansion, the *compound-list*
 1746 shall not be executed. Omitting

1747 `in word ...`

1748 is equivalent to

1749 `in "$@"`

1750 **Exit Status**

1751 The exit status of a `for` command shall be the exit status of the last command
 1752 that executes. If there are no items, the exit status shall be zero.

1753 **3.9.4.2.1 for Loop Rationale.** *(This subclause is not a part of P1003.2)*

1754 The format is shown with generous usage of <newline>s. See the grammar in
 1755 3.10 for a precise description of where <newline>s and semicolons can be inter-
 1756 changed.

1757 Some historical implementations support { and } as substitutes for `do` and `done`.
 1758 The working group chose to omit them, even as an obsolescent feature. (Note that
 1759 these substitutes were only for the `for` command; the `while` and `until` com- 1
 1760 mands could not use them historically, because they are followed by *compound-* 1
 1761 lists that may contain { . . . } grouping commands themselves 1

1762 The reserved word pair `do ... done` was selected rather than `do ... od` (which
 1763 would have matched the spirit of `if ... fi` and `case ... esac`) because `od` is a
 1764 commonly-used utility name and this would have been an unacceptable choice.

1765 **3.9.4.3 case Conditional Construct**

1766 The conditional construct `case` shall execute the *compound-list* corresponding to
 1767 the first one of several *patterns* (see 3.13) that is matched by the string resulting
 1768 from the tilde expansion, parameter expansion, command substitution, and arith-
 1769 metic expansion and quote removal of the given word. The reserved word `in`
 1770 shall denote the beginning of the patterns to be matched. Multiple patterns with
 1771 the same *compound-list* are delimited by the `|` symbol. The control operator `)` ter-
 1772 minates a list of patterns corresponding to a given action. The *compound-list* for
 1773 each list of patterns is terminated with `;;`. The `case` construct terminates with
 1774 the reserved word `esac` (case reversed).

1775 The format for the `case` construct is as follows.

```
1776     case word in
1777         [(pattern1)          compound-list ; ;           2
1778         [(pattern2|pattern3) compound-list ; ;           2
1779         ...
1780     esac
```

1781 The `;;` is optional for the last *compound-list*.

1782 Each pattern in a pattern list shall be expanded and compared against the expansion of *word*. After the first match, no more patterns shall be expanded, and the
1783 *compound-list* shall be executed. The order of expansion and comparing of patterns
1784 in a multiple pattern list is unspecified.
1785

1786 Exit Status

1787 The exit status of `case` is zero if no patterns are matched. Otherwise, the exit
1788 status shall be the exit status of the last command executed in the *compound-list*.

1789 3.9.4.3.1 case Conditional Construct Rationale. *(This subclause is not a part of* 1790 *P1003.2)*

1791 An optional open-parenthesis before *pattern* was added to allow numerous historical 2
1792 KornShell scripts to conform. At one time, using the leading parenthesis was 2
1793 required if the `case` statement were to be embedded within a `$ ()` command sub- 2
1794 stitution; this is no longer the case with the POSIX shell. Nevertheless, many 2
1795 existing scripts use the open-parenthesis, if only because it makes matching- 2
1796 parenthesis searching easier in `vi` and other editors. This is a relatively simple 2
1797 implementation change that is fully upward compatible for all scripts. 2

1798 Consideration was given to requiring `break` inside the *compound-list* to prevent
1799 falling through to the next pattern action list. This was rejected as being nonex-
1800 isting practice. An interesting undocumented feature of the KornShell is that
1801 using `&` instead of `;` as a terminator causes the exact opposite behavior—the
1802 flow of control continues with the next *compound-list*.

1803 The pattern `"*"`, given as the last pattern in a `case` construct, is equivalent to
1804 the default case in a C-language `switch` statement

1805 The grammar shows that reserved words can be used as patterns, even if one is
1806 the first word on a line. Obviously, the reserved word `esac` cannot be used in this
1807 manner.

1808 3.9.4.4 if Conditional Construct

1809 The `if` command shall execute a *compound-list* and use its exit status to deter-
1810 mine whether to execute another *compound-list*.

1811 The format for the `if` construct is as follows.

```
1812     if compound-list
1813     then
1814         compound-list
1815     [elif compound-list
1816     then
1817         compound-list] ...
1818     [else
1819         compound-list]
1820     fi
```

The *if compound-list* is executed; if its exit status is zero, the then *compound-list* is executed and the command shall complete. Otherwise, each *elif compound-list* is executed, in turn, and if its exit status is zero, the then *compound-list* is executed and the command shall complete. Otherwise, the *else compound-list* is executed.

Exit Status

The exit status of the *if* command shall be the exit status of the then or else *compound-list* that was executed, or zero, if none was executed.

3.9.4.4.1 *if* Conditional Construct Rationale. *(This subclause is not a part of P1003.2)*

The precise format for the command syntax is described in 3.10.

3.9.4.5 *while* Loop

The *while* loop continuously shall execute one *compound-list* as long as another *compound-list* has a zero exit status.

The format of the *while* loop is as follows

```
while compound-list-1
do
    compound-list-2
done
```

The *compound-list-1* shall be executed, and if it has a nonzero exit status, the *while* command shall complete. Otherwise, the *compound-list-2* shall be executed, and the process shall repeat.

Exit Status

The exit status of the *while* loop shall be the exit status of the last *compound-list-2* executed, or zero if none was executed.

3.9.4.5.1 *while* Loop Rationale. *(This subclause is not a part of P1003.2)*

The precise format for the command syntax is described in 3.10.

3.9.4.6 *until* Loop

The *until* loop continuously shall execute one *compound-list* as long as another *compound-list* has a nonzero exit status.

The format of the *until* loop is as follows

```
until compound-list-1
do
    compound-list-2
done
```

1856 The *compound-list-1* shall be executed, and if it has a zero exit status, the `until`
 1857 command shall complete. Otherwise, the *compound-list-2* shall be executed, and
 1858 the process shall repeat.

1859 **Exit Status**

1860 The exit status of the `until` loop shall be the exit status of the last *compound-*
 1861 *list-2* executed, or zero if none was executed.

1862 **3.9.4.6.1 `until` Loop Rationale.** *(This subclause is not a part of P1003.2)*

1863 The precise format for the command syntax is described in 3.10.

1864 **3.9.5 Function Definition Command**

1865 A function is a user-defined name that is used as a simple command to call a com-
 1866 pound command with new positional parameters. A function is defined with a
 1867 *function definition command*.

1868 The format of a function definition command is as follows:

1869 `fname () compound-command [io-redirect ...]`

1870 The function is named *fname*; it shall be a name (see 3.1.5). An implementation 1
 1871 may allow other characters in a function name as an extension. The implementa- 1
 1872 tion shall maintain separate namespaces for functions and variables.

1873 The argument *compound-command* represents a compound command, as
 1874 described in 3.9.4.

1875 When the function is declared, none of the expansions in 3.6 shall be performed
 1876 on the text in *compound-command* or *io-redirect*; all expansions shall be per-
 1877 formed as normal each time the function is called. Similarly, the optional *io-*
 1878 *redirect* redirections and any variable assignments within *compound-command*
 1879 shall be performed during the execution of the function itself, not the function
 1880 definition. See 3.8.1 for the consequences of failures of these operations on
 1881 interactive and noninteractive shells.

1882 When a function is executed, it shall have the syntax-error and variable-
 1883 assignment properties described for special built-in utilities, in the enumerated
 1884 list at the beginning of 3.14.

1885 The *compound-command* shall be executed whenever the function name is
 1886 specified as the name of a simple command (see 3.9.1.1). The operands to the
 1887 command temporarily shall become the positional parameters during the execu-
 1888 tion of the *compound-command*; the special parameter `#` shall also be changed to
 1889 reflect the number of operands. The special parameter `0` shall be unchanged.
 1890 When the function completes, the values of the positional parameters and the spe-
 1891 cial parameter `#` shall be restored to the values they had before the function was
 1892 executed. If the special built-in `return` is executed in the *compound-command*,
 1893 the function shall complete and execution shall resume with the next command
 1894 after the function call.

Exit Status

The exit status of a function definition shall be zero if the function was declared successfully; otherwise, it shall be greater than zero. The exit status of a function invocation shall be the exit status of the last command executed by the function.

3.9.5.1 Function Definition Command Rationale *(This subclause is not a part of P1003.2)*

The description of functions in Draft 8 was based on the notion that functions should behave like miniature shell scripts; that is, except for sharing variables, most elements of an execution environment should behave as if it were a new execution environment, and changes to these should be local to the function. For example, traps and options should be reset on entry to the function, and any changes to them don't affect the traps or options of the caller. There were numerous objections to this basic idea, and the opponents asserted that functions were intended to be a convenient mechanism for grouping commonly executed commands that were to be executed in the current execution environment, similar to the execution of the `dot` special built-in.

Opponents also pointed out that the functions described in Draft 8 did not scope everything a new shell script would anyway, such as the current working directory, or `umask`, but instead picked a few select properties. The basic argument was that if one wanted scoping of the execution environment, the mechanism already exists: put the commands in a new shell script and call it. All traditional shells that implemented functions, other than the KornShell, have implemented functions that operate in the current execution environment. Because of this, Draft 9 removed any local scoping of traps or options. Local variables within a function were considered and included in Draft 9 (controlled by the special built-in `local`), but were removed because they do not fit the simple model developed for the scoping of functions and there was some opposition to adding yet another new special built-in from outside existing practice. Implementations should reserve the identifier `local` (as well as `typeset`, as used in the KornShell) in case this local variable mechanism is adopted in a future version of POSIX.2.

A separate issue from the execution environment of a function is the availability of that function to child shells. A few objectors, including the author of the original Version 7 UNIX system shell, maintained that just as a variable can be shared with child shells by exporting it, so should a function—and so this capability has been added to the standard. In previous drafts, the `export` command therefore had a `-f` flag for exporting functions. Functions that were exported were to be put into the environment as `name()=value` pairs, and upon invocation, the shell would scan the environment for these, and automatically define these functions. This facility received a lot of balloting opposition and was removed from Draft 11. Some of the arguments against exportable functions were:

- There was little existing practice. The Ninth Edition shell provided them, but there was controversy over how well it worked.

1937 — There are numerous security problems associated with functions appearing
 1938 in a script's environment and overriding standard utilities or the
 1939 application's own utilities.

1940 — There was controversy over requiring `make` to import functions, where it
 1941 has historically used an `exec` function for many of its command line execu-
 1942 tions.

1943 — Functions can be big and the environment is of a limited size. (The
 1944 counter-argument was that functions are no different than variables in
 1945 terms of size: there can be big ones, and there can be small ones—and just
 1946 as one does not export huge variables, one does not export huge functions.
 1947 However, this insight might be lost on the average shell-function writer,
 1948 who typically writes much larger functions than variables.)

1949 As far as can be determined, the functions in POSIX.2 match those in System V.
 1950 The KornShell has two methods of defining functions:

```
1951     function fname { compound-list }
```

1952 and

```
1953     fname() { compound-list }
```

1954 The latter uses the same definition as POSIX.2, but differs in semantics, as
 1955 described previously. A future edition of the KornShell is planned to align the
 1956 latter syntax with POSIX and keep the former as-is.

1957 The name space for functions is limited to that of a *name* because of historical 1
 1958 practice. Complications in defining the syntactic rules for the function definition 1
 1959 command and in dealing with known extensions such as the KornShell's `@()` 1
 1960 prevented the name space from being widened to a *word*, as requested by some 1
 1961 balloters. Using functions to support synonyms such as the C-shell's `!!` and `%` is 1
 1962 thus disallowed to portable applications, but acceptable as an extension. For 1
 1963 interactive users, the aliasing facilities in the UPE should be adequate for this 1
 1964 purpose. It is recognized that the name space for utilities in the file system is 1
 1965 wider than that currently supported for functions, if the portable filename charac- 1
 1966 ter set guidelines are ignored, but it did not seem useful to mandate extensions in 1
 1967 systems for so little benefit to portable applications. 1

1968 The `()` in the function definition command consists of two operators. Therefore,
 1969 intermixing `<blank>`s with the *fname*, `(`, and `)` is allowed, but unnecessary.

1970 An example of how a function definition can be used wherever a simple command
 1971 is allowed:

```
1972     # If variable i is equal to "yes",  

1973     # define function foo to be ls -l  

1974     #  

1975     [ X$i = Xyes ] && foo() {  

1976         ls -l  

1977     }
```

3.10 Shell Grammar

The following grammar describes the Shell Command Language. Any discrepancies found between this grammar and the preceding description shall be resolved in favor of this clause.

3.10.1 Shell Grammar Lexical Conventions

The input language to the shell must be first recognized at the character level. The resulting tokens shall be classified by their immediate context according to the following rules (applied in order). These rules are used to determine what a “token” that is subject to parsing at the token level is. The rules for token recognition in 3.3 shall apply.

- (1) A <newline> shall be returned as the token identifier `NEWLINE`.
- (2) If the token is an operator, the token identifier for that operator shall result.
- (3) If the string consists solely of digits and the delimiter character is one of < or >, the token identifier `IO_NUMBER` shall be returned.
- (4) Otherwise, the token identifier `TOKEN` shall result.

Further distinction on `TOKEN` is context-dependent. It may be that the same `TOKEN` yields `WORD`, a `NAME`, an `ASSIGNMENT`, or one of the reserved words below, dependent upon the context. Some of the productions in the grammar below are annotated with a rule number from the following list. When a `TOKEN` is seen where one of those annotated productions could be used to reduce the symbol, the applicable rule shall be applied to convert the token identifier type of the `TOKEN` to a token identifier acceptable at that point in the grammar. The reduction shall then proceed based upon the token identifier type yielded by the rule applied. When more than one rule applies, the highest numbered rule shall apply (which in turn may refer to another rule). [Note that except in rule (7), the presence of an = in the token has no effect.]

The `WORD` tokens shall have the word expansion rules applied to them immediately before the associated command is executed, not at the time the command is parsed.

3.10.2 Shell Grammar Rules

- (1) [Command Name]
When the `TOKEN` is exactly a reserved word, the token identifier for that reserved word shall result. Otherwise, the token `WORD` shall be returned. 1
Also, if the parser is in any state where only a reserved word could be the next correct token, proceed as above. 1
- NOTE: Because at this point quote marks are retained in the token, quoted strings cannot be recognized as reserved words. This rule also implies that reserved words will not be recognized except in certain positions in the input, such as after a <newline> or

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2017	semicolon; the grammar presumes that if the reserved word is intended, it will be properly delimited by the user, and does not attempt to reflect that requirement directly. Also	
2018	note that line joining is done before tokenization, as described in 3.2.1, so escaped new-	
2019	lines are already removed at this point.	
2020		
2021	NOTE: Rule (1) is not directly referenced in the grammar, but is referred to by other	1
2022	rules, or applies globally.	1
2023	(2) [Redirection to/from filename]	
2024	The expansions specified in 3.7 shall occur. As specified there, exactly	
2025	one field can result (or the result is unspecified), and there are additional	1
2026	requirements on pathname expansion.	
2027	(3) [Redirection from here-document]	
2028	Quote removal [3.7.4]. shall be applied to the word to determine the del-	1
2029	imiter that will be used to find the end of the here-document that begins	1
2030	after the next <newline>.	1
2031	(4) [Case statement termination]	
2032	When the TOKEN is exactly the reserved word Esac, the token identifier	
2033	for Esac shall result. Otherwise, the token WORD shall be returned.	
2034	(5) [NAME in for]	
2035	When the TOKEN meets the requirements for a name [3.1.5], the token	
2036	identifier NAME shall result. Otherwise, the token WORD shall be returned.	
2037	(6) [Third word of for and case]	
2038	When the TOKEN is exactly the reserved word In, the token identifier for	
2039	In shall result. Otherwise, the token WORD shall be returned.	
2040	(7) [Assignment preceding command name]	1
2041	(a) [When the first word]	
2042	If the TOKEN does not contain the character =, rule (1) shall be	
2043	applied. Otherwise, apply (7)(b).	
2044	(b) [Not the first word]	
2045	If the TOKEN contains the equals-sign character:	
2046	— If it begins with =, the token WORD shall be returned.	
2047	— If all the characters preceding = form a valid name [3.1.5], the	
2048	token ASSIGNMENT_WORD shall be returned. (Quoted characters	
2049	cannot participate in forming a valid name.)	
2050	— Otherwise, it is unspecified whether it is ASSIGNMENT_WORD or	
2051	WORD that is returned.	
2052	Assignment to the NAME shall occur as specified in 3.9.1.	
2053	(8) [NAME in function]	
2054	When the TOKEN is exactly a reserved word, the token identifier for that	
2055	reserved word shall result. Otherwise, when the TOKEN meets the	
2056	requirements for a name [3.1.5], the token identifier NAME shall result.	
2057	Otherwise, rule (7) shall apply.	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

```

2058     (9) [Body of function]
2059     Word expansion and assignment shall never occur, even when required
2060     by the rules above, when this rule is being parsed. Each TOKEN that
2061     might either be expanded or have assignment applied to it shall instead
2062     be returned as a single WORD consisting only of characters that are exactly
2063     the token described in 3.3.

2064     /* -----
2065     The grammar symbols
2066     ----- */

2067     %token WORD
2068     %token ASSIGNMENT_WORD
2069     %token NAME
2070     %token NEWLINE
2071     %token IO_NUMBER

2072     /* The following are the operators mentioned above. */
2073     %token AND_IF OR_IF DSEMI
2074     /* '&&' '||' ';;' */
2075     %token DLESS DGREAT LESSAND GREATAND LESSGREAT DLESSDASH
2076     /* '<<' '>>' '<&' '>&' '<>' '<<-' */
2077     %token CLOBBER
2078     /* '>|' */
2079     /* The following are the reserved words */
2080     %token If Then Else Elif Fi Do Done
2081     /* 'if' 'then' 'else' 'elif' 'fi' 'do' 'done' */
2082     %token Case Esac While Until For
2083     /* 'case' 'esac' 'while' 'until' 'for' */
2084     /* These are reserved words, not operator tokens, and are
2085     recognized when reserved words are recognized. */
2086     %token Lbrace Rbrace Bang
2087     /* '{' '}' '!' */
2088     %token In
2089     /* 'in' */
2090     /* -----
2091     The Grammar
2092     ----- */
2093     %start complete_command
2094     %%

2095     complete_command : list separator
2096                     | list
2097                     ;
2098     list              : list separator_op and_or
2099                     | and_or
2100                     ;
2101     and_or            : pipeline

```

1

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```

2102      | and_or AND_IF linebreak pipeline
2103      | and_or OR_IF  linebreak pipeline
2104      ;
2105 pipeline      :      pipe_sequence
2106      | Bang pipe_sequence
2107      ;
2108 pipe_sequence  :      command
2109      | pipe_sequence '|' linebreak command
2110      ;
2111 command        : simple_command
2112      | compound_command
2113      | compound_command redirect_list
2114      | function_definition
2115      ;
2116 compound_command : brace_group
2117      | subshell
2118      | for_clause
2119      | case_clause
2120      | if_clause
2121      | while_clause
2122      | until_clause
2123      ;
2124 subshell       : '(' compound_list ')'
2125      ;
2126 compound_list  :      term
2127      | newline_list term
2128      |      term separator
2129      | newline_list term separator
2130      ;
2131 term           : term separator and_or
2132      |      and_or
2133      ;
2134 for_clause     : For name do_group
2135      | For name In wordlist sequential_sep do_group
2136      ;
2137 name           : NAME /* Apply rule (5) */
2138      ;
2139 in             : In /* Apply rule (6) */
2140      ;
2141 wordlist       : wordlist WORD
2142      |      WORD
2143      ;
2144 case_clause     : Case WORD In linebreak case_list Esac
2145      | Case WORD In linebreak      Esac
2146      ;
2147 case_list      : case_list case_item
2148      |      case_item
2149      ;

```

2

```

2150 case_item      :      pattern ')' linebreak      DSEMI linebreak
2151                |      pattern ')' compound_list DSEMI linebreak
2152                | '(' pattern ')' linebreak      DSEMI linebreak      2
2153                | '(' pattern ')' compound_list DSEMI linebreak      2
2154                ;
2155 pattern         :      WORD      /* Apply rule (4) */
2156                | pattern '|' WORD      /* Do not apply rule (4) */      1
2157                ;
2158 if_clause       : If compound_list Then compound_list else_part Fi
2159                | If compound_list Then compound_list      Fi
2160                ;
2161 else_part       : Elif compound_list Then else_part
2162                | Else compound_list
2163                ;
2164 while_clause    : While compound_list do_group
2165                ;
2166 until_clause    : Until compound_list do_group
2167                ;
2168 function_definition : fname '(' ')' linebreak function_body
2169                ;
2170 function_body   : compound_command      /* Apply rule (9) */
2171                | compound_command redirect_list /* Apply rule (9) */
2172                ;
2173 fname          : NAME      /* Apply rule (8) */      2
2174                ;
2175 brace_group     : Lbrace compound_list Rbrace
2176                ;
2177 do_group        : Do compound_list Done
2178                ;
2179 simple_command  : cmd_prefix cmd_word cmd_suffix
2180                | cmd_prefix cmd_word
2181                | cmd_prefix
2182                | cmd_name cmd_suffix
2183                | cmd_name
2184                ;
2185 cmd_name        : WORD      /* Apply rule (7)(a) */
2186                ;
2187 cmd_word        : WORD      /* Apply rule (7)(b) */
2188                ;
2189 cmd_prefix      :      io_redirect
2190                | cmd_prefix io_redirect
2191                |      ASSIGNMENT_WORD
2192                | cmd_prefix ASSIGNMENT_WORD
2193                ;
2194 cmd_suffix      :      io_redirect
2195                | cmd_suffix io_redirect
2196                |      WORD

```

```

2197         | cmd_suffix WORD
2198         ;

2199 redirect_list :          io_redirect
2200         | redirect_list io_redirect
2201         ;

2202 io_redirect :          io_file
2203         | IO_NUMBER io_file
2204         |          io_here
2205         | IO_NUMBER io_here
2206         ;

2207 io_file : '<'      filename
2208         | LESSAND  filename
2209         | '>'      filename
2210         | GREATAND filename
2211         | DGREAT   filename
2212         | LESSGREAT filename
2213         | CLOBBER  filename
2214         ;

2215 filename : WORD /* Apply rule (2) */
2216         ;

2217 io_here : DLESS      here_end
2218         | DLESSDASH here_end
2219         ;

2220 here_end : WORD /* Apply rule (3) */
2221         ;

2222 newline_list :          NEWLINE
2223         | newline_list NEWLINE
2224         ;

2225 linebreak : newline_list
2226         | /* empty */
2227         ;

2228 separator_op : '&'
2229         | ';'
2230         ;

2231 separator : separator_op linebreak
2232         | newline_list
2233         ;

2234 sequential_sep : ';' linebreak
2235         | newline_list
2236         ;

```

3.10.3 Shell Grammar Rationale. *(This subclause is not a part of P1003.2)*

There are several subtle aspects of this grammar where conventional usage implies rules about the grammar that in fact are not true.

For `compound_list`, only the forms that end in a separator allow a reserved word to be recognized, so usually only a separator can be used where a compound list precedes a reserved word (such as `Then`, `Else`, `Do`, and `Rbrace`. Explicitly requiring a separator would disallow such valid (if rare) statements as:

```
if (false) then (echo x) else (echo y) fi
```

See the NOTE under special grammar rule (1).

Concerning the third sentence of rule (1) (“Also, if the parser ...”):

- This sentence applies rather narrowly: when a compound list is terminated by some clear delimiter (such as the closing `fi` of an inner `if_clause`) then it would apply; where the compound list might continue (as in after a `;`), rule (7a) [and consequently the first sentence of rule (1)] would apply. In many instances the two conditions are identical, but this part of rule (1) does not give license to treating a `WORD` as a reserved words unless it is in a place where a reserved word must appear.

- The statement is equivalent to requiring that when the LR(1) lookahead set contains exactly a reserved word, it must be recognized if it is present. (Here “LR(1)” refers to the theoretical concepts, not to any real parser generator.)

For example, in the construct below, and when the parser is at the point marked with `^`, the only next legal token is `then` (this follows directly from the grammar rules).

```
if if....fi then .... fi
      ^
```

At that point, the `then` must be recognized as a reserved word.

(Depending on the actual parser generator actually used, “extra” reserved words may be in some lookahead sets. It does not really matter if they are recognized, or even if any possible reserved word is recognized in that state, because if it is recognized and is not in the (theoretical) LR(1) lookahead set, an error will ultimately be detected. In the example above, if some other reserved word (e.g., `while`) is also recognized, an error will occur later.

This is approximately equivalent to saying that reserved words are recognized after other reserved words (because it is after a reserved word that this condition will occur), but avoids the “except for...” list that would be required for `case`, `for`, etc. (Reserved words are of course recognized anywhere a `simple_command` can appear, as well. Other rules take care of the special cases of nonrecognition, such as rule (4) for `case` statements.)

Note that the body of here-documents are handled by Token Recognition (see 3.3) and do not appear in the grammar directly. (However, the here-document I/O

2279 redirection operator is handled as part of the grammar.)

2280 The start symbol of the grammar (`complete_command`) represents either input
 2281 from the command line or a shell script. It is repeatedly applied by the inter-
 2282 preter to its input, and represents a single “chunk” of that input as seen by the
 2283 interpreter. 1

2284 The processing of here-documents is handled as part of token recognition (see 3.3)
 2285 rather than as part of the grammar.

2286 3.11 Signals and Error Handling

2287 When a command is in an asynchronous list, the shell shall prevent SIGQUIT and
 2288 SIGINT signals from the keyboard from interrupting the command. Otherwise,
 2289 signals shall have the values inherited by the shell from its parent (see also
 2290 3.14.13).

2291 When a signal for which a trap has been set is received while the shell is waiting 1
 2292 for the completion of a utility executing a foreground command, the trap associ- 1
 2293 ated with that signal shall not be executed until after the foreground command 1
 2294 has completed. When the shell is waiting, by means of the `wait` utility, for asyn- 1
 2295 chronous commands to complete, the reception of a signal for which a trap has 1
 2296 been set shall cause the `wait` utility to return immediately with an exit status 1
 2297 >128, immediately after which the trap associated with that signal shall be taken. 1

2298 If multiple signals are pending for the shell for which there are associated trap
 2299 actions (see 3.14.13), the order of execution of trap actions is unspecified.

2300 3.12 Shell Execution Environment

2301 A shell execution environment consists of the following:

- 2302 — Open files inherited upon invocation of the shell, plus open files controlled
 2303 by `exec`.
- 2304 — Working Directory as set by `cd` (see 4.5).
- 2305 — File Creation Mask set by `umask` (see 4.67).
- 2306 — Current traps set by `trap` (see 3.14.13).
- 2307 — Shell parameters that are set by variable assignment (see `set` in 3.14.11)
 2308 or from the POSIX.1 {8} environment inherited by the shell when it begins
 2309 (see `export` in 3.14.8).
- 2310 — Shell functions (see 3.9.5.)
- 2311 — Options turned on at invocation or by `set`.
- 2312 — Process IDs of the last commands in asynchronous lists known to this shell 1
 2313 environment; see 3.9.3.1. 1

Utilities other than the special built-ins (see 3.14) shall be invoked in a separate environment that consists of the following. The initial value of these objects shall be the same as that for the parent shell, except as noted below.

- Open files inherited on invocation of the shell, open files controlled by the `exec` special built-in (see 3.14.6), plus any modifications and additions specified by any redirections to the utility.
- Current working directory.
- File creation mask.
- If the utility is a shell script, traps caught by the shell shall be set to the default values and traps ignored by the shell shall be set to be ignored by the utility. If the utility is not a shell script, the trap actions (default or ignore) shall be mapped into the appropriate signal handling actions for the utility.
- Variables with the `export` attribute, along with those explicitly exported for the duration of the command, shall be passed to the utility as POSIX.1 {8} environment variables.

The environment of the shell process shall not be changed by the utility unless explicitly specified by the utility description (for example, `cd` and `umask`).

A subshell environment shall be created as a duplicate of the shell environment, except that signal traps set by that shell environment shall be set to the default values. Changes made to the subshell environment shall not affect the shell environment. Command substitution, commands that are grouped with parentheses, and asynchronous lists shall be executed in a subshell environment. Additionally, each command of a multicommand pipeline is in a subshell environment; as an extension, however, any or all commands in a pipeline may be executed in the current environment. All other commands shall be executed in the current shell environment.

3.12.0.1 Shell Execution Environment Rationale. *(This subclause is not a part of P1003.2)*

Some systems have implemented the last stage of a pipeline in the current environment so that commands such as

```
command | read foo
```

set variable `foo` in the current environment. It was decided to allow this extension, but not require it; therefore, a shell programmer should consider a pipeline to be in a subshell environment, but not depend on it.

The previous description of execution environment failed to mention that each command in a multiple command pipeline could be in a subshell execution environment. For compatibility with some existing shells, the wording was phrased to allow an implementation to place any or all commands of a pipeline in the current environment. However, this means that a POSIX application must assume each command is in a subshell environment, but not depend on it.

2355 The wording about shell scripts is meant to convey the fact that describing “trap
 2356 actions” can only be understood in the context of the shell command language.
 2357 Outside this context, such as in a C-language program, signals are the operative
 2358 condition, not traps.

2359 3.13 Pattern Matching Notation

2360 The pattern matching notation described in this clause is used to specify patterns
 2361 for matching strings in the shell. Historically, pattern matching notation is
 2362 related to, but slightly different from, the regular expression notation described in
 2363 2.8. For this reason, the description of the rules for this pattern matching nota-
 2364 tion are based on the description of regular expression notation.

2365 3.13.0.1 Pattern Matching Notation Rationale. *(This subclause is not a part of* 2366 *P1003.2)*

2367 Pattern matching is a simpler concept and has a simpler syntax than regular
 2368 expressions, as the former is generally used for the manipulation of file names,
 2369 which are relatively simple collections of characters, while the latter is generally
 2370 used to manipulate arbitrary text strings of potentially greater complexity. How-
 2371 ever, some of the basic concepts are the same, so this clause points liberally to the
 2372 detailed descriptions in 2.8.

2373 3.13.1 Patterns Matching a Single Character

2374 The following *patterns matching a single-character* match a single character:
 2375 *ordinary characters*, *special pattern characters*, and *pattern bracket expressions*.
 2376 The pattern bracket expression also shall match a single collating element.

2377 An ordinary character is a pattern that shall match itself. It can be any character
 2378 in the supported character set except for NUL, those special shell characters in 3.2 1
 2379 that require quoting, and the following three special pattern characters. Match- 1
 2380 ing shall be based on the bit pattern used for encoding the character, not on the 1
 2381 graphic representation of the character. If any character (ordinary, shell special, 1
 2382 or pattern special) is quoted, that pattern shall match the character itself. The 1
 2383 shell special characters always require quoting. 1

2384 When unquoted and outside a bracket expression, the following three characters 1
 2385 shall have special meaning in the specification of patterns: 1

- 2386 ? A question-mark is a pattern that shall match any character.
- 2387 * An asterisk is a pattern that shall match multiple characters, as
 2388 described in 3.13.2.
- 2389 [The open bracket shall introduce a pattern bracket expression.

2390 The description of basic regular expression bracket expressions in 2.8.3.2 also
 2391 shall apply to the pattern bracket expression, except that the exclamation-mark

character (!) shall replace the circumflex character (^) in its role in a *nonmatching list* in the regular expression notation. A bracket expression starting with an unquoted circumflex character produces unspecified results.

When pattern matching is used where shell quote removal is not performed [such as in the argument to the `find -name` primary when `find` is being called using an `exec` function, or in the *pattern* argument to the `fnmatch()` function], special characters can be escaped to remove their special meaning by preceding them with a <backslash>. This escaping <backslash> shall be discarded. The sequence `\\` shall represent one literal backslash. All of the requirements and effects of quoting on ordinary, shell special, and special pattern characters shall apply to escaping in this context.

3.13.1.1 Patterns Matching a Single Character Rationale. *(This subclause is not a part of P1003.2)*

Both “quoting” and “escaping” are described here because pattern matching must work in three separate circumstances:

- Calling directly upon the shell, such as in pathname expansion or in a case statement. All of the following will match the string or file `abc`: `abc`, `"abc"`, `a"b"c`, `a\bc`, `a[b]c`, `a["b"]c`, `a[\b]c`, `a?c`, `a*c`. The following will not: `"a?c"`, `a*c`, `a\[b]c`, `a["\b"]c`.
- Calling a utility or function without going through a shell, as described for `find` and `fnmatch()`.
- Calling utilities such as `find` or `pax` through the shell command line. (Although `find` and `pax` are the only instances of this in the standard utilities, describing it globally here is useful for future utilities that may use pattern matching internally.) In this case, shell quote removal is performed before the utility sees the argument. For example, in

```
find /bin -name "e\c[\h]o" -print
```

after quote removal, the backslashes are presented to `find` and it treats them as escape characters. Both precede ordinary characters, so the `c` and `h` represent themselves and `echo` would be found on many historical systems (that have it in `/bin`). To find a filename that contained shell special characters or pattern characters, both quoting and escaping are required, such as

```
pax -r ... "a\(\?"
```

to extract a filename ending with “a(?”.

Conforming applications are required to quote or escape the shell special characters (called “metacharacters” in some historical documentation). If used without this protection, syntax errors can result or implementation extensions can be triggered. For example, the KornShell supports a series of extensions based on parentheses in patterns.

2432 The restriction on circumflex in a bracket expression is to allow implementations
 2433 that support pattern matching using circumflex as the negation character in addi-
 2434 tion to the exclamation-mark.

2435 3.13.2 Patterns Matching Multiple Characters

2436 The following rules are used to construct *patterns matching multiple characters*
 2437 from *patterns matching a single character*:

- 2438 (1) The asterisk (*) is a pattern that shall match any string, including the
 2439 null string.
- 2440 (2) The concatenation of *patterns matching a single character* is a valid pat-
 2441 tern that shall match the concatenation of the single characters or collat-
 2442 ing elements matched by each of the concatenated patterns.
- 2443 (3) The concatenation of one or more *patterns matching a single character*
 2444 with one or more asterisks is a valid pattern. In such patterns, each
 2445 asterisk shall match a string of zero or more characters, matching the
 2446 greatest possible number of characters that still allows the remainder of
 2447 the pattern to match the string.

2448 3.13.2.1 Patterns Matching Multiple Characters Rationale. *(This subclause is* 2449 *not a part of P1003.2)*

2450 Since each asterisk matches “zero or more” occurrences, the patterns `a*b` and
 2451 `a**b` have identical functionality.

2452 *Examples:*

- 2453 `a[bc]` matches the strings `ab` and `ac`.
- 2454 `a*d` matches the strings `ad`, `abd`, and `abcd`, but not the string `abc`.
- 2455 `a*d*` matches the strings `ad`, `abcd`, `abcdef`, `aaaad`, and `adddd`;
- 2456 `*a*d` matches the strings `ad`, `abcd`, `efabcd`, `aaaad`, and `adddd`.

2457 3.13.3 Patterns Used for Filename Expansion

2458 The rules described so far in 3.13.1 and 3.13.2 are qualified by the following rules
 2459 that apply when pattern matching notation is used for filename expansion.

- 2460 (1) The slash character in a pathname shall be explicitly matched by using
 2461 one or more slashes in the pattern; it cannot be matched by the asterisk
 2462 or question-mark special characters or by a bracket expression. Slashes
 2463 in the pattern are identified before bracket expressions; thus, a slash
 2464 cannot be included in a pattern bracket expression used for filename
 2465 expansion.
- 2466 (2) If a filename begins with a period (.), the period shall be explicitly
 2467 matched by using a period as the first character of the pattern or

2468 immediately following a slash character. The leading period shall not be
2469 matched by:

- 2470 — The asterisk or question-mark special characters, or
- 2471 — A bracket expression containing a nonmatching list (such as `[!a]`), a
2472 range expression (such as `[%-0]`), or a character class expression
2473 (such as `[[:punct:]]`).

2474 It is unspecified whether an explicit period in a bracket expression
2475 matching list (such as `[.abc]`) can match a leading period in a filename.

- 2476 (3) Specified patterns are matched against existing filenames and path-
2477 names, as appropriate. Each component that contains a pattern character 2
2478 requires read permission in the directory containing that component. 2
2479 Any component that does not contain a pattern character requires search 2
2480 permission. For example, given the pattern 2

2481 `/foo/bar/x*/bam` 2

2482 search permission is needed for directory `/foo`, search and read permis- 2
2483 sions are needed for directory `bar`, and search permission is needed for 2
2484 each `x*` directory. If the pattern matches any existing filenames or path-
2485 names, the pattern shall be replaced with those filenames and path-
2486 names, sorted according to the collating sequence in effect in the current
2487 locale. If the pattern contains an invalid bracket expression or does not
2488 match any existing filenames or pathnames, the pattern string shall be
2489 left unchanged.

2490 **3.13.3.1 Patterns Used for File Name Expansion Rationale.** *(This subclause is* 2491 *not a part of P1003.2)*

2492 The caveat about a slash within a bracket expression is derived from historical
2493 practice. The pattern `a[b/c]d` will not match such pathnames as `abd` or `a/d`. It
2494 will only match a pathname of literally `a[b/c]d`.

2495 Filenames beginning with a period historically have been specially protected from
2496 view on UNIX systems. A proposal to allow an explicit period in a bracket expres-
2497 sion to match a leading period was considered; it is allowed as an implementation
2498 extension, but a conforming application cannot make use of it. If this extension
2499 becomes popular in the future, it will be considered for a future version of
2500 POSIX.2.

2501 Historical systems have varied in their permissions requirements. To match 2
2502 `f*/bar` has required read permissions on the `f*` directories in the System V shell, 2
2503 but this standard, the C-shell, and KornShell require only search permissions. 2

2504 3.14 Special Built-in Utilities

2505 The following *special built-in* utilities shall be supported in the shell command
 2506 language. The output of each command, if any, shall be written to standard out-
 2507 put, subject to the normal redirection and piping possible with all commands.

2508 The term *built-in* implies that the shell can execute the utility directly and does
 2509 not need to search for it. An implementation can choose to make any utility a
 2510 built-in; however, the special built-in utilities described here differ from regular
 2511 built-in utilities in two respects:

- 2512 (1) A syntax error in a special built-in utility may cause a shell executing
 2513 that utility to abort, while a syntax error in a regular built-in utility shall
 2514 not cause a shell executing that utility to abort. (See 3.8.1 for the conse-
 2515 quences of errors on interactive and noninteractive shells.) If a special
 2516 built-in utility encountering a syntax error does not abort the shell, its
 2517 exit value shall be nonzero.
- 2518 (2) Variable assignments specified with special built-in utilities shall remain
 2519 in effect after the built-in completes; this shall not be the case with a reg- 1
 2520 ular built-in or other utility. 1

2521 As described in 2.3, the special built-in utilities in this clause need not be pro-
 2522 vided in a manner accessible via the POSIX.1 {8} *exec* family of functions.

2523 Some of the special built-ins are described as conforming to the utility argument
 2524 syntax guidelines in 2.10.2. For those that are not, the requirement in 2.11.3 that
 2525 "--" be recognized as a first argument to be discarded does not apply and a con-
 2526 forming application shall not use that argument.

2527 3.14.1 break — Exit from for, while, or until loop

2528 break [*n*]

2529 Exit from the smallest enclosing *for*, *while*, or *until* loop, if any; or from the
 2530 *n*th enclosing loop if *n* is specified. The value of *n* is an unsigned decimal integer 1
 2531 ≥ 1 . The default is equivalent to $n=1$. If *n* is greater than the number of enclosing
 2532 loops, the last enclosing loop shall be exited from. Execution continues with the
 2533 command immediately following the loop.

2534 Exit Status

- | | | | |
|------|----|---|---|
| 2535 | 0 | Successful completion. | 2 |
| 2536 | >0 | The <i>n</i> value was not an unsigned decimal integer ≥ 1 . | 2 |

2537 **3.14.1.1 break Rationale.** *(This subclause is not a part of P1003.2)*

2538 **Example:**

```
2539     for i in *
2540     do
2541         if test -d "$i"
2542         then break
2543         fi
2544     done
```

2545 Consideration was given to expanding the syntax of the `break` and `continue` to refer to a label associated with the appropriate loop, as a preferable alternative to the `[n]` method. This new method was proposed late in the development of the standard and adequate consensus could not be formed to include it. However, POSIX.2 does reserve the namespace of command names ending with a colon. It is anticipated that a future implementation could take advantage of this and provide something like:

```
2552     outofloop: for i in a b c d e
2553     do
2554         for j in 0 1 2 3 4 5 6 7 8 9
2555         do
2556             if test -r "${i}${j}"
2557             then break outofloop
2558             fi
2559         done
2560     done
```

1

2561 and that this might be standardized after implementation experience is achieved.

2562 **3.14.2 colon — Null utility**

2563 `:` [*argument ...*]

2564 This utility shall only expand command *arguments*.

2565 **Exit Status**

2566 Zero.

2567 **3.14.2.1 colon Rationale.** *(This subclause is not a part of P1003.2)*

2568 The colon (`:`), or null utility, is used when a command is needed, as in the `then` condition of an `if` command, but nothing is to be done by the command.

2570 **Example:**


```

2571      : ${X=abc}
2572      if      false
2573      then    :
2574      else    echo $X
2575      fi
2576      abc

```

2577 As with any of the special built-ins, the null utility can also have variable assign-
 2578 ments and redirections associated with it, such as:

```

2579      x=y : > z

```

2580 which sets variable *x* to the value *y* (so that it persists after the null utility “com-
 2581 pletes”) and creates or truncates file *z*.

2582 3.14.3 **continue** — Continue for, while, or until loop

```

2583      continue [n]

```

2584 The **continue** utility shall return to the top of the smallest enclosing **for**,
 2585 **while**, or **until**, loop, or to the top of the *n*th enclosing loop, if *n* is specified.
 2586 This involves repeating the condition list of a **while** or **until** loop or performing
 2587 the next assignment of a **for** loop, and reexecuting the loop if appropriate.

2588 The value of *n* is a decimal integer ≥ 1 . The default is equivalent to *n*=1. If *n* is
 2589 greater than the number of enclosing loops, the last enclosing loop is used.

2590 **Exit Status**

2591	0	Successful completion.	2
2592	>0	The <i>n</i> value was not an unsigned decimal integer ≥ 1 .	2

2593 3.14.3.1 **continue Rationale.** (*This subclause is not a part of P1003.2*)

2594 **Example:**

```

2595      for i in *
2596      do
2597          if test -d "$i"
2598          then continue
2599          fi
2600      done

```

2601 3.14.4 **dot** — Execute commands in current environment

```

2602      . file

```

2603 The shell shall execute commands from the *file* in the current environment.

2604 If *file* does not contain a slash, the shell shall use the search path specified by
 2605 **PATH** to find the directory containing *file*. Unlike normal command search, how-
 2606 ever, the file searched for by the **dot** utility need not be executable. If no

2607 readable file is found, a noninteractive shell shall abort; an interactive shell shall
 2608 write a diagnostic message to standard error, but this condition shall not be con-
 2609 sidered a syntax error.

2610 **Exit Status**

2611 Returns the value of the last command executed, or a zero exit status if no com-
 2612 mand is executed.

2613 **3.14.4.1 dot Rationale.** *(This subclause is not a part of P1003.2)*

2614 Some older implementations searched the current directory for the *file*, even if the
 2615 value of **PATH** disallowed it. This behavior was omitted from POSIX.2 due to con-
 2616 cerns about introducing the susceptibility to trojan horses that the user might be
 2617 trying to avoid by leaving dot out of **PATH**.

2618 The KornShell version of `dot` takes optional arguments that are set to the posi- 1
 2619 tional parameters. This is a valid extension that allows a dot script to behave 1
 2620 identically to a function.

2621 **Example:**

```
2622     cat foobar
2623     foo=hello bar=world
2624     . foobar
2625     echo $foo $bar
2626     hello world
```

2627 **3.14.5 eval — Construct command by concatenating arguments**

```
2628     eval [argument ...]
```

2629 The `eval` utility shall construct a command by concatenating *arguments*
 2630 together, separating each with a <space>. The constructed command shall be
 2631 read and executed by the shell.

2632 **Exit Status**

2633 If there are no *arguments*, or only null arguments, `eval` shall return a zero exit
 2634 status; otherwise, it shall return the exit status of the command defined by the
 2635 string of concatenated *arguments* separated by spaces.

3.14.5.1 `eval` Rationale. *(This subclause is not a part of P1003.2)*

Example:

```
foo=10 x=foo
y=' '$x
echo $y
$foo
eval y=' '$x
echo $y
10
```

3.14.6 `exec` — Execute commands and open, close, and/or copy file descriptors

```
exec [command [argument ...]]
```

The `exec` utility opens, closes, and/or copies file descriptors as specified by any redirections as part of the command.

If `exec` is specified without *command* or *arguments*, and any file descriptors with numbers > 2 are opened with associated redirection statements, it is unspecified whether those file descriptors remain open when the shell invokes another utility.

If `exec` is specified with *command*, it shall replace the shell with *command* without creating a new process. If *arguments* are specified, they are arguments to *command*. Redirection shall affect the current shell execution environment.

Exit Status

If *command* is specified, `exec` shall not return to the shell; rather, the exit status of the process shall be the exit status of the program implementing *command*, which overlaid the shell. If *command* is not found, the exit status shall be 127. If *command* is found, but it is not an executable utility, the exit status shall be 126. If a redirection error occurs (see 3.8.1), the shell shall exit with a value in the range 1–125. Otherwise, `exec` shall return a zero exit status.

3.14.6.1 `exec` Rationale. *(This subclause is not a part of P1003.2)*

Most historical implementations are not conformant in that

```
foo=bar exec cmd
```

does not pass `foo` to `cmd`.

Earlier drafts stated that “If specified without *command* or *argument*, the shell sets to close-on-exec file numbers greater than 2 that are opened in this way, so that they will be closed when the shell invokes another program.” This was based on the behavior of one version of the KornShell and was made unspecified when it was realized that some existing scripts relied on the more generally historical behavior (leaving all file descriptors open). Furthermore, since the application should have no cognizance of whether a new shell is simply *fork*(ed), rather than

2674 *exec()*ed, it could not consistently rely on the automatic closing behavior anyway.
 2675 Scripts concerned that child shells could misuse open file descriptors can always
 2676 close them explicitly, as shown in one of the following examples.

2677 **Examples:**

2678 Open readfile as file descriptor 3 for reading:

2679 `exec 3< readfile`

2680 Open writefile as file descriptor 4 for writing:

2681 `exec 4> writefile`

2682 Make unit 5 a copy of unit 0:

2683 `exec 5<&0`

2684 Close file unit 3:

2685 `exec 3<&-`

2686 Cat the file maggie by replacing the current shell with the `cat` utility:

2687 `exec cat maggie`

2688 **3.14.7 exit — Cause the shell to exit**

2689 `exit [n]`

2690 The `exit` utility shall cause the shell to exit with the exit status specified by the 1
 2691 unsigned decimal integer *n*. If *n* is specified, but its value is not between 0 and 1
 2692 255 inclusively, the exit status is undefined. 1

2693 A trap on `EXIT` shall be executed before the shell terminates, except when the
 2694 `exit` utility is invoked in that trap itself, in which case the shell shall exit
 2695 immediately.

2696 **Exit Status**

2697 The exit status shall be *n*, if specified. Otherwise, the value shall be the exit
 2698 value of the last command executed, or zero if no command was executed. When
 2699 `exit` is executed in a trap action (see 3.14.13), the “last command” is considered
 2700 to be the command that executed immediately preceding the trap action.

2701 **3.14.7.1 exit Rationale.** *(This subclause is not a part of P1003.2)*

2702 As explained in other clauses, certain exit status values have been reserved for 1
 2703 special uses and should be used by applications only for those purposes: 1

2704 126 A file to be executed was found, but it was not an executable utility. 1

2705 127 A utility to be executed was not found. 1

2706 >128 A command was interrupted by a signal. 1

Examples:Exit with a *true* value:

```
exit 0
```

Exit with a *false* value:

```
exit 1
```

3.14.8 export — Set export attribute for variables

```
export name[=word]...
```

```
export -p
```

The shell shall give the export attribute to the variables corresponding to the specified *names*, which shall cause them to be in the environment of subsequently executed commands.

When `-p` is specified, `export` shall write to the standard output the names and values of all exported variables, in the following format:

```
"export %s=%s\n", <name>, <value>
```

The shell shall format the output, including the proper use of quoting, so that it is suitable for re-input to the shell as commands that achieve the same exporting results.

The `export` special built-in shall conform to the utility argument syntax guidelines described in 2.10.2.

Exit Status

Zero.

3.14.8.1 export Rationale. *(This subclause is not a part of P1003.2)*

When no arguments are given, the results are unspecified. Some historical shells use the no-argument case as the functional equivalent of what is required here with `-p`. This feature was left unspecified because it is not existing practice in all shells and some scripts may rely on the now-unspecified results on their implementations. Attempts to specify the `-p` output as the default case were unsuccessful in achieving consensus. The `-p` option was added to allow portable access to the values that can be saved and then later restored using, for instance, a dot script.

Examples:Export **PWD** and **HOME** variables:

```
export PWD HOME
```

Set and export the **PATH** variable:

2741 export PATH=/local/bin:\$PATH

2742 **Save and restore all exported variables:**

2743 export -p > *temp-file*
 2744 unset *a lot of variables*
 2745 . . . *processing*
 2746 . *temp-file*

2747 **3.14.9 readonly — Set read-only attribute for variables**

1

2748 readonly *name*[=*word*]...
 2749 readonly -p

2750 The variables whose *names* are specified shall be given the readonly attribute.
 2751 The values of variables with the read-only attribute cannot be changed by subse-
 2752 quent assignment, nor can those variables be unset by the unset utility.

2753 When -p is specified, readonly shall write to the standard output the names and
 2754 values of all read-only variables, in the following format:

1

2755 "readonly %s=%s\n", <*name*>, <*value*>

2756 The shell shall format the output, including the proper use of quoting, so that it is
 2757 suitable for re-input to the shell as commands that achieve the same attribute-
 2758 setting results.

2759 The readonly special built-in shall conform to the utility argument syntax guide-
 2760 lines described in 2.10.2.

2761 **Exit Status**

2762 Zero.

2763 **3.14.9.1 readonly Rationale.** (*This subclause is not a part of P1003.2*)

2764 **Example:**

2765 readonly HOME PWD

2766 Some versions of the shell exist that preserve the read-only attribute across
 2767 separate invocations. POSIX.2 allows this behavior, but does not require it.

2768 See the rationale for export (3.14.8.1) for a description of the no-argument and
 2769 -p output cases.

2770 In a previous draft, read-only functions were considered, but they were omitted as
 2771 not being existing practice or particularly useful. Furthermore, functions must
 2772 not be readonly across invocations to preclude *spoofing* (spoofing is the term for
 2773 the practice of creating a program that acts like a well-known utility with the
 2774 intent of subverting the user's real intent) of administrative or security-relevant
 2775 (or -conscious) shell scripts.

2776 3.14.10 **return** — Return from a function

2777 `return [n]`

2778 The `return` utility shall cause the shell to stop executing the current function or
2779 dot script (see 3.14.4). If the shell is not currently executing a function or dot
2780 script, the results are unspecified.

2781 **Exit Status**

2782 The value of the special parameter `?` shall be set to *n*, an unsigned decimal
2783 integer, or to the exit status of the last command executed if *n* is not specified. If
2784 the value of *n* is greater than 255, the results are undefined. When `return` is
2785 executed in a trap action (see 3.14.13), the “last command” is considered to be the
2786 command that executed immediately preceding the trap action.

2787 3.14.10.1 **return Rationale.** (*This subclause is not a part of P1003.2*)

2788 The behavior of `return` when not in a function or dot script differs between the
2789 System V shell and the KornShell. In the System V shell this is an error,
2790 whereas in the KornShell, the effect is the same as `exit`.

2791 The results of returning a number greater than 255 are undefined because of
2792 differing practices in the various historical implementations. Some shells AND
2793 out all but the low order 8 bits; others allow larger values, but not of unlimited
2794 size.

2795 See the discussion of appropriate exit status values in 3.14.7.1.

1

2796 3.14.11 **set** — Set/unset options and positional parameters

2797 `set [-aCefnuvx] [argument ...]`

2798 `set [+aCefnuvx] [argument ...]`

2799 `set -- [argument ...]`

2800 *Obsolescent version:*

2801 `set - [argument ...]`

2802 If no options or *arguments* are specified, `set` shall write the names and values of
2803 all shell variables in the collation sequence of the current locale. Each *name* shall
2804 start on a separate line, using the format:

2805 `"%s=%s\n", <name>, <value>`

2806 The *value* string shall be written with appropriate quoting so that it is suitable
2807 for re-input to the shell, (re)setting, as far as possible, the variables that are
2808 currently set. Readonly variables cannot be reset. See the description of shell
2809 quoting in 3.2.

1

1

2810 When options are specified, they shall set or unset attributes of the shell, as
2811 described below. When *arguments* are specified, they shall cause positional
2812 parameters to be set or unset, as described below. Setting/unsetting attributes

2813 and positional parameters are not necessarily related actions, but they can be
 2814 combined in a single invocation of `set`.

2815 The `set` utility shall conform to the utility argument syntax guidelines described
 2816 in 2.10.2, except that options can be specified with either a leading hyphen (mean-
 2817 ing enable the option) or plus-sign (meaning disable it).

2818 The implementation shall support the options in the following list in both their
 2819 hyphen and plus-sign forms. These options can also be specified as options to `sh`;
 2820 see 4.56.

2821	-a	When this option is on, the export attribute shall be set for each variable to which an assignment is performed. (See 3.1.15.) If the assignment precedes a utility name in a command, the export attributes shall not persist in the current execution environment after the utility completes, with the exception that preceding one of the special built-in utilities shall cause the export attribute to persist after the built-in has completed. If the assignment does not precede a utility name in the command, or if the assignment is a result of the operation of the <code>getopts</code> or <code>read</code> utilities (see 4.27 and 4.52), the export attribute shall persist until the variable is unset.	1 1 1 1 1 1 1 1 1 1
2832	-C	(Uppercase C.) Prevent existing files from being overwritten by the shell's <code>></code> redirection operator (see 3.7.2); the <code>> </code> redirection operator shall override this “noclobber” option for an individual file.	
2836	-e	When this option is on, if a simple command fails for any of the reasons listed in 3.8.1 or returns an exit status value <code>>0</code> , and is not part of the compound list following a <code>while</code> , <code>until</code> , or <code>if</code> keyword, and is not a part of an AND or OR list, and is not a pipe- line preceded by the <code>!</code> reserved word, then the shell immediately shall exit.	1 1 1 1 1
2842	-f	The shell shall disable pathname expansion.	
2843	-n	The shell shall read commands but not execute them; this can be used to check for shell script syntax errors. An interactive shell may ignore this option.	
2846	-u	The shell shall write a message to standard error when it tries to expand a variable that is not set and immediately exit. An interactive shell shall not exit.	
2849	-v	The shell shall write its input to standard error as it is read.	
2850	-x	The shell shall write to standard error a trace for each command after it expands the command and before it executes it.	

2852 The default for all these options is off (unset) unless the shell was invoked with
 2853 them on (see `sh` in 4.56). All the positional parameters shall be unset before any
 2854 new values are assigned.

The remaining arguments shall be assigned in order to the positional parameters. The special parameter # shall be set to reflect the number of positional parameters.

The special argument "--" immediately following the `set` command name can be used to delimit the arguments if the first argument begins with + or -, or to prevent inadvertent listing of all shell variables when there are no arguments. The command `set --` without *arguments* shall unset all positional parameters and set the special parameter # to zero.

In the obsolescent version, the `set` command name followed by - with no other arguments shall turn off the -v and -x options without changing the positional parameters. The `set` command name followed by - with other arguments shall turn off the -v and -x options and assign the arguments to the positional parameters in order.

Exit Status

Zero.

3.14.11.1 `set` Rationale. *(This subclause is not a part of P1003.2)*

The `set --` form is listed specifically in the Synopsis even though this usage is implied by the utility syntax guidelines. The explanation of this feature removes any ambiguity about whether the `set --` form might be misinterpreted as being equivalent to `set` without any options or arguments. The functionality of this form has been adopted from the KornShell. In System V, `set --` only unsets parameters if there is at least one argument; the only way to unset all parameters is to use `shift`. Using the KornShell version should not affect System V scripts because there should be no reason to deliberately issue it without arguments; if it were issued as, say:

```
set -- "$@"
```

and there were in fact no arguments resulting from `$@`, unsetting the parameters would be a no-op anyway.

The `set +` form in earlier drafts was omitted as being an unnecessary duplication of `set` alone and not widespread historical practice.

The `noclobber` option was changed to -C from the `set -o noclobber` option in previous drafts. The `set -o` is used in the KornShell to accept word-length option names, duplicating many of the single-letter names. The `noclobber` option was changed to a single letter so that the historical `$-` paradigm would not be broken; see 3.5.2.

The following `set` flags were intentionally omitted with the following rationale:

- h This flag is related to command name hashing, which is not required for an implementation. It is primarily a performance issue, which is outside the scope of this standard.

2894 -k The -k flag was originally added by Bourne to make it easier for users of
 2895 prerelease versions of the shell. In early versions of the Bourne shell
 2896 the construct `set name=value`, had to be used to assign values to shell
 2897 variables. The problem with -k is that the behavior affects parsing, vir-
 2898 tually precluding writing any compilers. To explain the behavior of -k,
 2899 it is necessary to describe the parsing algorithm, which is implementa-
 2900 tion defined. For example,

2901 `set -k; echo name=value`

2902 and

2903 `set -k`
 2904 `echo name=value`

2905 behave differently. The interaction with functions is even more com-
 2906 plex. What is more, the -k flag is never needed, since the command line
 2907 could have been reordered.

2908 -t The -t flag is hard to specify and almost never used. The only known
 2909 use could be done with here-documents. Moreover, the behavior with
 2910 ksh and sh differ. The man page says that it exits after reading and
 2911 executing one command. What is one command? If the input is
 2912 `date;date`, sh executes both `date` commands, ksh does only the first.

2913 Consideration was given to rewriting `set` to simplify its confusing syntax. A
 2914 specific suggestion was that the `unset` utility should be used to unset options
 2915 instead of using the non-*getopt*(-)-able *+option* syntax. However, the conclusion
 2916 was reached that people were satisfied with the existing practice of using *+option*
 2917 and there was no compelling reason to modify such widespread existing practice.

2918 **Examples:**

2919 Write out all variables and their values:

2920 `set`

2921 Set \$1, \$2, and \$3 and set \$# to 3:

2922 `set c a b`

2923 Turn on the -x and -v options:

2924 `set -xv`

2925 Unset all positional parameters:

2926 `set --`

2927 Set \$1 to the value of x, even if x begins with - or +:

2928 `set -- "$x"`

2929 Set the positional parameters to the expansion of x, even if x expands with a lead-
 2930 ing - or +:

2931 `set -- $x`

3.14.12 shift — Shift positional parameters

```
shift [n]
```

The positional parameters shall be shifted. Positional parameter 1 shall be assigned the value of parameter (1+*n*), parameter 2 shall be assigned the value of parameter (2+*n*), and so forth. The parameters represented by the numbers \$# down to \$#-*n*+1 shall be unset, and the parameter # shall be updated to reflect the new number of positional parameters.

The value *n* shall be an unsigned decimal integer less than or equal to the value of the special parameter #. If *n* is not given, it shall be assumed to be 1. If *n* is 0, the positional and special parameters shall not be changed.

Exit Status

The exit status shall be >0 if *n*>#; otherwise, it shall be zero.

3.14.12.1 shift Rationale. *(This subclause is not a part of P1003.2)***Example:**

```
set a b c d e
shift 2
echo $*
c d e
```

3.14.13 trap — Trap signals

```
trap [action condition ...]
```

If *action* is -, the shell shall reset each *condition* to the default value. If *action* is null (' '), the shell shall ignore each of the specified *conditions* if they arise. Otherwise, the argument *action* shall be read and executed by the shell when one of the corresponding conditions arises. The action of the trap shall override a previous action (either default action or one explicitly set). The value of \$? after the trap action completes shall be the value it had before the trap was invoked.

The condition can be EXIT, 0 (equivalent to EXIT), or a signal specified using a symbolic name, without the SIG prefix, as listed in Required Signals and Job Control Signals (Table 3-1 and Table 3-2 in POSIX.1 {8}). (For example: HUP, INT, QUIT, TERM). Setting a trap for SIGKILL or SIGSTOP produces undefined results.

The environment in which the shell executes a trap on EXIT shall be identical to the environment immediately after the last command executed before the trap on EXIT was taken.

Each time the trap is invoked, the *action* argument shall be processed in a manner equivalent to:

```
eval "$action"
```

Signals that were ignored on entry to a noninteractive shell cannot be trapped or reset, although no error need be reported when attempting to do so. An interactive shell may reset or catch signals ignored on entry. Traps shall remain in place for a given shell until explicitly changed with another `trap` command.

The `trap` command with no arguments shall write to standard output a list of commands associated with each condition. The format is:

```
"trap -- %s %s...\n", <action>, <condition> ...
```

The shell shall format the output, including the proper use of quoting, so that it is suitable for re-input to the shell as commands that achieve the same trapping results.

An implementation may allow numeric signal numbers for the conditions as an extension, if and only if the following map of signal numbers to names is true:

<u>Signal Number</u>	<u>Signal Name</u>	<u>Signal Number</u>	<u>Signal Name</u>
1	SIGHUP	9	SIGKILL
2	SIGINT	14	SIGALRM
3	SIGQUIT	15	SIGTERM
6	SIGABRT		

Otherwise, it shall be an error for the application to use numeric signal numbers.

The `trap` special built-in shall conform to the utility argument syntax guidelines described in 2.10.2.

Exit Status

If the `trap` name or number is invalid, a nonzero exit status shall be returned; otherwise, zero shall be returned. For both interactive and noninteractive shells, invalid signal names or numbers shall not be considered a syntax error and shall not cause the shell to abort.

3.14.13.1 `trap` Rationale. *(This subclause is not a part of P1003.2)*

Implementations may permit lowercase signal names as an extension. Implementations may also accept the names with the `SIG` prefix; no known historical shell does so. The `trap` and `kill` utilities in POSIX.2 are now consistent in their omission of the `SIG` prefix for signal names. Some `kill` implementations do not allow the prefix and `kill -l` lists the signals without prefixes.

As stated previously, when a subshell is entered, traps are set to the default actions. This does not imply that the `trap` command cannot be used within the subshell to set new traps.

Trapping `SIGKILL` or `SIGSTOP` is accepted by some historical implementations, but it does not work. Portable POSIX.2 applications cannot try it.

The output format is not historical practice. Since the output of historical `traps` is not portable (because numeric signal values are not portable) and had to

3007 change to become so, an opportunity was taken to format the output in a way that
 3008 a shell script could use to save and then later reuse a trap if it wanted. For exam-
 3009 ple:

```
3010     save_traps=$(trap)
3011     . . .
3012     eval "$save_traps"
```

3013 The KornShell uses an `ERR` trap that is triggered whenever `set -e` would cause
 3014 an exit. This is allowable as an extension, but was not mandated, as other shells
 3015 have not used it.

3016 The text about the environment for the `EXIT` trap invalidates the behavior of some
 3017 historical versions of interactive shells which, e.g., close the standard input before
 3018 executing a trap on 0. For example, in some historical interactive shell sessions
 3019 the following trap on 0 would always print `--`:

```
3020     trap 'read foo; echo "-$foo-"' 0
```

3021 **Examples:**

3022 Write out a list of all traps and actions:

```
3023     trap
```

3024 Set a trap so the `logout` utility in the **HOME** directory will execute when the
 3025 shell terminates:

```
3026     trap '$HOME/logout' EXIT
3027     or
3028     trap '$HOME/logout' 0
```

3029 Unset traps on `INT`, `QUIT`, `TERM`, and `EXIT`:

```
3030     trap - INT QUIT TERM EXIT
```

3031 **3.14.14 unset — Unset values and attributes of variables and functions**

```
3032     unset [-fv] name ... 1
```

3033 Each variable or function specified by *name* shall be unset.

3034 If `-v` is specified, *name* refers to a variable name and the shell shall unset it and 1
 3035 remove it from the environment. Read-only variables cannot be unset. 1

3036 If `-f` is specified, *name* refers to a function and the shell shall unset the function 1
 3037 definition. 1

3038 If neither `-f` nor `-v` is specified, *name* refers to a variable; if a variable by that 1
 3039 name does not exist, it is unspecified whether a function by that name, if any, 1
 3040 shall be unset. 1

3041 Unsetting a variable or function that was not previously set shall not be con-
 3042 sidered an error and shall not cause the shell to abort. 1

3043 The `unset` special built-in shall conform to the utility argument syntax guide-
 3044 lines described in 2.10.2.

3045 **Exit Status**

3046 0 All *names* were successfully unset.

3047 >0 At least one *name* could not be unset.

3048 **3.14.14.1 `unset` Rationale.** *(This subclause is not a part of P1003.2)*

3049 Note that

3050 VARIABLE=

3051 is not equivalent to an `unset` of `VARIABLE`; in the example, `VARIABLE` is set to `" "`.
 3052 Also, the “variables” that can be `unset` should not be misinterpreted to include
 3053 the special parameters (see 3.5.2).

3054 Consideration was given to omitting the `-f` option in favor of an `unfunction` util-
 3055 ity, but decided to retain existing practice.

3056 The `-v` option was introduced because System V historically used one name space 1
 3057 for both variables and functions. When `unset` is used without options, System V 1
 3058 historically `unset` either a function or a variable and there was no confusion about 1
 3059 which one was intended. A portable POSIX.2 application can use `unset` without 1
 3060 an option to `unset` a variable, but not a function; the `-f` option must be used. 1

3061 **Examples:**

3062 Unset the **VISUAL** variable:

3063 `unset -v VISUAL` 1

3064 Unset the functions `foo` and `bar`:

3065 `unset -f foo bar`

Section 4: Execution Environment Utilities

The Execution Environment Utilities are the utilities that shall be implemented in all conforming POSIX.2 systems.

4.1 **awk** — Pattern scanning and processing language

4.1.1 Synopsis

```
awk [-F ERE] [-v assignment] ... program [argument ... ]
```

```
awk [-F ERE] -f progfile ... [-v assignment] ... [argument ... ]
```

4.1.2 Description

The **awk** utility shall execute programs written in the *awk* programming language, which is specialized for textual data manipulation. An **awk** program is a sequence of patterns and corresponding actions. When input is read that matches a pattern, the action associated with that pattern shall be carried out.

Input shall be interpreted as a sequence of records. By default, a record is a line, but this can be changed by using the `RS` built-in variable. Each record of input shall be matched in turn against each pattern in the program. For each pattern matched, the associated action shall be executed.

The **awk** utility shall interpret each input record as a sequence of fields where, by default, a field is a string of non-`<blank>` characters. This default white space field delimiter can be changed by using the `FS` built-in variable or the `-F ERE`. The **awk** utility shall denote the first field in a record `$1`, the second `$2`, and so forth. The symbol `$0` shall refer to the entire record; setting any other field shall cause the reevaluation of `$0`. Assigning to `$0` shall reset the values of all other fields and the `NF` built-in variable.

1
1

4.1.3 Options

The `awk` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

- `-F ERE` Define the input field separator to be the extended regular expression *ERE*, before any input is read (see 4.1.7.4).
- `-f progfile` Specifies the pathname of the file *progfile* containing an `awk` program. If multiple instances of this option are specified, the concatenation of the files specified as *progfile* in the order specified shall be the `awk` program. The `awk` program can alternatively be specified in the command line as a single argument.
- `-v assignment`
The *assignment* argument shall be in the same form as an *assignment* operand. The specified variable assignment shall occur prior to executing the `awk` program, including the actions associated with `BEGIN` patterns (if any). Multiple occurrences of this option can be specified.

4.1.4 Operands

The following operands shall be supported by the implementation:

- program* If no `-f` option is specified, the first operand to `awk` shall be the text of the `awk` program. The application shall supply the *program* operand as a single argument to `awk`. If the text does not end in a `<newline>` character, `awk` shall interpret the text as if it did.
- argument* Either of the following two types of *arguments* can be intermixed:
 - file* A pathname of a file that contains the input to be read, which is matched against the set of patterns in the program. If no *file* operands are specified, or if a *file* operand is `-`, the standard input shall be used.
 - assignment*
An operand that begins with an underscore or alphabetic character from the portable character set (see Table 2-3 in 2.4), followed by a sequence of underscores, digits, and alphabets from the portable character set, followed by the `=` character shall specify a variable assignment rather than a pathname. The characters before the `=` shall represent the name of an `awk` variable; if that name is an `awk` reserved word (see 4.1.7.7) the behavior is undefined. The characters following the equals-sign shall be interpreted as if they appeared in the `awk` program preceded and followed by a double-

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

quote (") character, as a `STRING` token (see 4.1.7.7), except that if the last character is an unescaped backslash, it shall be interpreted as a literal backslash rather than as the first character of the sequence "\". The variable shall be assigned the value of that `STRING` token. If that value is considered a *numeric string* (see 4.1.7.2), the variable shall also be assigned its numeric value. Each such variable assignment shall occur just prior to the processing of the following *file*, if any. Thus, an assignment before the first *file* argument shall be executed after the `BEGIN` actions (if any), while an assignment after the last *file* argument shall occur before the `END` actions (if any). If there are no *file* arguments, assignments shall be executed before processing the standard input.

4.1.5 External Influences

4.1.5.1 Standard Input

The standard input shall be used only if no *file* operands are specified, or if a *file* operand is `-`. See Input Files.

4.1.5.2 Input Files

Input files to the `awk` program from any of the following sources: 1

- Any *file* operands or their equivalents, achieved by modifying the `awk` variables `ARGV` and `ARGC` 1
- Standard input in the absence of any *file* operands 1
- Arguments to the `getline` function 1

shall be text files. Whether the variable `RS` is set to a value other than `<newline>` or not, for these files, the implementation shall support records terminated with the specified separator up to `{LINE_MAX}` bytes and may support longer records. 1

If `-f progfile` is specified, the file(s) named by *progfile* shall be text file(s) containing an `awk` program.

4.1.5.3 Environment Variables

The following environment variables shall affect the execution of `awk`:

LANG This variable shall determine the locale to use for the locale categories when both `LC_ALL` and the corresponding environment variable (beginning with `LC_`) do not specify a locale. See 2.6.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

101	LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
102		
103		
104		
105	LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files), the behavior of character classes within regular expressions, the identification of characters as letters, and the mapping of upper- and lowercase characters for the <code>toupper</code> and <code>tolower</code> functions.
106		
107		
108		
109		
110		
111		
112	LC_COLLATE	This variable shall determine the locale for the behavior of ranges, equivalence classes, and multicharacter collating elements within regular expressions and in comparisons of string values.
113		
114		
115		
116	LC_MESSAGES	This variable shall determine the language in which messages should be written.
117		
118	LC_NUMERIC	This variable shall determine the radix character used when interpreting numeric input, performing conversions between numeric and string values, and formatting numeric output.
119		
120		
121		
122	PATH	This variable shall define the search path when looking for commands executed by <code>system(<i>expr</i>)</code> , or input and output pipes. See 2.6.
123		
124		
125	In addition, all environment variables shall be visible via the <code>awk</code> variable <code>ENVIRON</code> .	
126		

127 **4.1.5.4 Asynchronous Events**

128 Default.

129 **4.1.6 External Effects**

130 **4.1.6.1 Standard Output**

131 The nature of the output files depends on the `awk` program.

132 **4.1.6.2 Standard Error**

133 Used only for diagnostic messages.

4.1.6.3 Output Files

The nature of the output files depends on the `awk` program.

4.1.7 Extended Description

4.1.7.1 Overall Program Structure

An `awk` program is composed of pairs of the form:

```
pattern { action }
```

Either the pattern or the action (including the enclosing brace characters) can be omitted.

A missing pattern shall match any record of input, and a missing action shall be equivalent to an action that writes the matched record of input to standard output.

Execution of the `awk` program shall start by first executing the actions associated with all `BEGIN` patterns in the order they occur in the program. Then each *file* operand (or standard input if no files were specified) shall be processed in turn by reading data from the file until a record separator is seen (`<newline>` by default), splitting the current record into fields using the current value of `FS` according to the rules in 4.1.7.4, evaluating each pattern in the program in the order of occurrence, and executing the action associated with each pattern that matches the current record. The action for a matching pattern shall be executed before evaluating subsequent patterns. Last, the actions associated with all `END` patterns shall be executed in the order they occur in the program.

4.1.7.2 Expressions

Expressions describe computations used in *patterns* and *actions*. In Table 4-1, valid expression operations are given in groups from highest precedence first to lowest precedence last, with equal-precedence operators grouped between horizontal lines. In expression evaluation, higher precedence operators shall be evaluated before lower precedence operators. In this table *expr*, *expr1*, *expr2*, and *expr3* represent any expression, while *lvalue* represents any entity that can be assigned to (i.e., on the left side of an assignment operator). The precise syntax of expressions is given in the grammar in 4.1.7.7.

Each expression shall have either a string value, a numeric value, or both. Except as stated for specific contexts, the value of an expression shall be implicitly converted to the type needed for the context in which it is used. A string value shall be converted to a numeric value by the equivalent of the following calls to functions defined by the C Standard {7}:

```
setlocale(LC_NUMERIC, "");
numeric_value = atof(string_value);
```

A numeric value that is exactly equal to the value of an integer (see 2.9.2.1) shall

Table 4-1 – awk Expressions in Decreasing Precedence

Syntax	Name	Semantic Definition	Type of Result	Assoc
(<i>expr</i>)	Grouping	C Standard {7}	type of <i>expr</i>	n/a
<i>\$expr</i>	Field reference	4.1.7.2	string	n/a
<i>++ lvalue</i>	Pre-increment	C Standard {7}	numeric	n/a
<i>-- lvalue</i>	Pre-decrement	C Standard {7}	numeric	n/a
<i>lvalue ++</i>	Post-increment	C Standard {7}	numeric	n/a
<i>lvalue --</i>	Post-decrement	C Standard {7}	numeric	n/a
<i>expr ^ expr</i>	Exponentiation	4.1.7.2	numeric	right
<i>! expr</i>	Logical not	C Standard {7}	numeric	n/a
<i>+ expr</i>	Unary plus	C Standard {7}	numeric	n/a
<i>- expr</i>	Unary minus	C Standard {7}	numeric	n/a
<i>expr * expr</i>	Multiplication	C Standard {7}	numeric	left
<i>expr / expr</i>	Division	C Standard {7}	numeric	left
<i>expr % expr</i>	Modulus	4.1.7.2	numeric	left
<i>expr + expr</i>	Addition	C Standard {7}	numeric	left
<i>expr - expr</i>	Subtraction	C Standard {7}	numeric	left
<i>expr expr</i>	String concatenation	4.1.7.2	string	left
<i>expr < expr</i>	Less than	4.1.7.2	numeric	none
<i>expr <= expr</i>	Less than or equal to	4.1.7.2	numeric	none
<i>expr != expr</i>	Not equal to	4.1.7.2	numeric	none
<i>expr == expr</i>	Equal to	4.1.7.2	numeric	none
<i>expr > expr</i>	Greater than	4.1.7.2	numeric	none
<i>expr >= expr</i>	Greater than or equal to	4.1.7.2	numeric	none
<i>expr ~ expr</i>	ERE match	4.1.7.4	numeric	none
<i>expr !~ expr</i>	ERE nonmatch	4.1.7.4	numeric	none
<i>expr in array</i>	Array membership	4.1.7.2	numeric	left
(<i>index</i>) in <i>array</i>	Multidimension array membership	4.1.7.2	numeric	left
<i>expr && expr</i>	Logical AND	C Standard {7}	numeric	left
<i>expr expr</i>	Logical OR	C Standard {7}	numeric	left
<i>expr1 ? expr2</i> : <i>expr3</i>	Conditional expression	C Standard {7}	type of selected <i>expr2</i> or <i>expr3</i>	right
<i>lvalue ^ = expr</i>	Exponentiation assignment	4.1.7.2	numeric	right
<i>lvalue % = expr</i>	Modulus assignment	4.1.7.2	numeric	right
<i>lvalue * = expr</i>	Multiplication assignment	C Standard {7}	numeric	right
<i>lvalue / = expr</i>	Division assignment	C Standard {7}	numeric	right
<i>lvalue + = expr</i>	Addition assignment	C Standard {7}	numeric	right
<i>lvalue - = expr</i>	Subtraction assignment	C Standard {7}	numeric	right
<i>lvalue = expr</i>	Assignment	C Standard {7}	type of <i>expr</i>	right

1
1
1

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

be converted to a string by the equivalent of a call to the `sprintf` function (see 4.1.7.6.2) with the string `"%d"` as the *fmt* argument and the numeric value being converted as the first and only *expr* argument. Any other numeric value shall be converted to a string by the equivalent of a call to the `sprintf` function with the value of the variable `CONVFMT` as the *fmt* argument and the numeric value being converted as the first and only *expr* argument. The result of the conversion is unspecified if the value of `CONVFMT` is not a floating-point format specification. This standard specifies no explicit conversions between numbers and strings. An application can force an expression to be treated as a number by adding zero to it, or can force it to be treated as a string by concatenating the null string (`" "`) to it.

A string value shall be considered to be a *numeric string* in the following case:

- (1) Any leading and trailing `<blank>`s shall be ignored.
- (2) If the first unignored character is a `+` or `-`, it shall be ignored.
- (3) If the remaining unignored characters would be lexically recognized as a `NUMBER` token (as described by the lexical conventions in 4.1.7.7), the string shall be considered a *numeric string*.

If a `-` character is ignored in the above steps, the numeric value of the *numeric string* shall be the negation of the numeric value of the recognized `NUMBER` token. Otherwise the numeric value of the *numeric string* shall be the numeric value of the recognized `NUMBER` token. Whether or not a string is a *numeric string* shall be relevant only in contexts where that term is used in this clause.

When an expression is used in a Boolean context (the first subexpression of a conditional expression, an expression operated on by logical NOT, logical AND, or logical OR, the second expression of a `for` statement, the expression of an `if` statement, or the expression of a `while` statement), if it has a numeric value, a value of zero shall be treated as false and any other value shall be treated as true. Otherwise, a string value of the null string shall be treated as false and any other value shall be treated as true.

All arithmetic shall follow the semantics of floating point arithmetic as specified by the C Standard [7]; see 2.9.2.

The value of the expression

$$expr1 \wedge expr2$$

shall be equivalent to the value returned by the C Standard [7] function call

$$\text{pow}(expr1, expr2)$$

The expression

$$lvalue \wedge= expr$$

shall be equivalent to the C Standard [7] expression

$$lvalue = \text{pow}(lvalue, expr)$$

except that *lvalue* shall be evaluated only once. The value of the expression

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

255 *expr1 % expr2*

256 shall be equivalent to the value returned by the C Standard {7} function call

257 *fmod(expr1, expr2)*

258 The expression

259 *lvalue %= expr*

260 shall be equivalent to the C Standard {7} expression

261 *lvalue = fmod(lvalue, expr)*

262 except that *lvalue* shall be evaluated only once.

263 Variables and fields shall be set by the assignment statement:

264 *lvalue = expression*

265 and the type of *expression* shall determine the resulting variable type. The
 266 assignment includes the arithmetic assignments (*+=*, *-=*, **=*, */=*, *%=*, *^=*, *++*,
 267 *--*) all of which produce a numeric result. The left-hand side of an assignment
 268 and the target of increment and decrement operators can be one of a variable, an
 269 array with index, or a field selector.

270 The awk language shall supply arrays that are used for storing numbers or
 271 strings. Arrays need not be declared. They shall initially be empty, and their
 272 sizes shall change dynamically. The subscripts, or element identifiers, are
 273 strings, providing a type of associative array capability. An array name followed
 274 by a subscript within square brackets can be used as an *lvalue* and thus as an
 275 expression, as described in the grammar (see 4.1.7.7). Unsubscripted array
 276 names can be used in only the following contexts:

- 277 — A parameter in a function definition or function call.
- 278 — The NAME token following any use of the keyword *in* as specified in the
- 279 grammar (see 4.1.7.7). If the name used in this context is not an array
- 280 name, the behavior is undefined.

281 A valid array *index* shall consist of one or more comma-separated expressions,
 282 similar to the way in which multidimensional arrays are indexed in some pro-
 283 gramming languages. Because awk arrays are really one dimensional, such a
 284 comma-separated list shall be converted to a single string by concatenating the
 285 string values of the separate expressions, each separated from the other by the
 286 value of the SUBSEP variable. Thus, the following two index operations shall be
 287 equivalent:

288 *var[expr1, expr2, ..., exprn]*

289 *var[expr1 SUBSEP expr2 SUBSEP ... SUBSEP exprn]*

290 A multidimensioned *index* used with the *in* operator shall be parenthesized. The
 291 *in* operator, which tests for the existence of a particular array element, shall not
 292 cause that element to exist. Any other reference to a nonexistent array element
 293 shall automatically create it.

Comparisons (with the `<`, `<=`, `!=`, `==`, `>`, and `>=` operators) shall be made numerically if both operands are numeric or if one is numeric and the other has a string value that is a numeric string. Otherwise, operands shall be converted to strings as required and a string comparison shall be made using the locale-specific collation sequence. The value of the comparison expression shall be 1 if the relation is true, or 0 if the relation is false.

4.1.7.3 Variables and Special Variables

Variables can be used in an `awk` program by referencing them. With the exception of function parameters (see 4.1.7.6.2), they are not explicitly declared. Uninitialized scalar variables and array elements have both a numeric value of zero and a string value of the empty string.

Field variables shall be designated by a `$` followed by a number or numerical expression. The effect of the field number *expression* evaluating to anything other than a nonnegative integer is unspecified; uninitialized variables or string values need not be converted to numeric values in this context. New field variables can be created by assigning a value to them. References to nonexistent fields (i.e., fields after `$NF`), shall produce the null string. However, assigning to a nonexistent field [e.g., `$(NF+2) = 5`] shall increase the value of `NF`, create any intervening fields with the null string as their values, and cause the value of `$0` to be recomputed, with the fields being separated by the value of `OFS`. Each field variable shall have a string value when created. If the string, with any occurrence of the decimal-point character from the current locale changed to a `<period>`, would be considered a *numeric string* (see 4.1.7.2), the field variable shall also have the numeric value of the *numeric string*.

The implementation shall support the following other special variables that are set by `awk`:

ARGC	The number of elements in the ARGV array.
ARGV	An array of command line arguments, excluding options and the <i>program</i> argument, numbered from zero to ARGC-1.
	The arguments in ARGV can be modified or added to; ARGC can be altered. As each input file ends, <code>awk</code> shall treat the next nonnull element of ARGV, up through the current value of ARGC-1, as the name of the next input file. Thus, setting an element of ARGV to null means that it shall not be treated as an input file. The name <code>'-'</code> shall indicate the standard input. If an argument matches the format of an <i>assignment</i> operand, this argument shall be treated as an assignment rather than a <i>file</i> argument.
CONVFMT	The <code>printf</code> format for converting numbers to strings (except for output statements, where OFMT is used); <code>"%.6g"</code> by default.
ENVIRON	The variable ENVIRON is an array representing the value of the environment, as described in POSIX.1 {8} 2.7. The indices of the array shall be strings consisting of the names of the environment variables, and the value of each array element shall be a string

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

337		consisting of the value of that variable. If the value of an environ-
338		ment variable is considered a <i>numeric string</i> (see 4.1.7.2), the
339		array element shall also have its numeric value.
340		In all cases where the behavior of <code>awk</code> is affected by environment
341		variables [including the environment of any command(s) that <code>awk</code>
342		executes via the <code>system</code> function or via pipeline redirections with
343		the <code>print</code> statement, the <code>printf</code> statement, or the <code>getline</code>
344		function], the environment used shall be the environment at the
345		time <code>awk</code> began executing; it is implementation defined whether 1
346		any modification of <code>ENVIRON</code> affects this environment. 1
347	FILENAME	A pathname of the current input file. Inside a <code>BEGIN</code> action the
348		value is undefined. Inside an <code>END</code> action the value is the name of
349		the last input file processed.
350	FNR	The ordinal number of the current record in the current file.
351		Inside a <code>BEGIN</code> action the value is zero. Inside an <code>END</code> action the
352		value is the number of the last record processed in the last file
353		processed.
354	FS	Input field separator regular expression; <code><space></code> by default.
355	NF	The number of fields in the current record. Inside a <code>BEGIN</code> action,
356		the use of <code>NF</code> is undefined unless a <code>getline</code> function without a
357		<code>var</code> argument is executed previously. Inside an <code>END</code> action, <code>NF</code>
358		shall retain the value it had for the last record read, unless a sub-
359		sequent, redirected, <code>getline</code> function without a <code>var</code> argument is
360		performed prior to entering the <code>END</code> action.
361	NR	The ordinal number of the current record from the start of input.
362		Inside a <code>BEGIN</code> action the value is zero. Inside an <code>END</code> action the
363		value is the number of the last record processed.
364	OFMT	The <code>printf</code> format for converting numbers to strings in output
365		statements (see 4.1.7.6.1); <code>"%.6g"</code> by default. The result of the 2
366		conversion is unspecified if the value of <code>OFMT</code> is not a floating- 2
367		point format specification. 2
368	OFS	The <code>print</code> statement output field separation; <code><space></code> by
369		default.
370	ORS	The <code>print</code> statement output record separator; <code><newline></code> by
371		default.
372	RLENGTH	The length of the string matched by the <code>match</code> function.
373	RS	The first character of the string value of <code>RS</code> is the input record
374		separator; <code><newline></code> by default. If <code>RS</code> contains more than one
375		character, the results are unspecified. If <code>RS</code> is null, then records
376		are separated by sequences of one or more blank lines, leading or
377		trailing blank lines do not result in empty records at the begin-
378		ning or end of the input, and <code><newline></code> is always a field separa-
379		tor, no matter what the value of <code>FS</code> is.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

380 RSTART The starting position of the string matched by the `match` func-
 381 tion, numbering from 1. This is always equivalent to the return
 382 value of the `match` function.

383 SUBSEP The subscript separator string for multidimensional arrays; the
 384 default value is implementation defined.

385 4.1.7.4 Regular Expressions

386 The `awk` utility shall make use of the extended regular expression notation (see
 387 2.8.4) except that it shall allow the use of C-language conventions for escaping
 388 special characters within the EREs, as specified in Table 2-15 and Table 4-2; these 1
 389 escape sequences shall be recognized both inside and outside bracket expressions. 1
 390 Note that records need not be separated by `<newline>`s and string constants can 1
 391 contain `<newline>`s, so even the `\n` sequence is valid in `awk` EREs. Using a slash 1
 392 character within the regular expression requires the escaping shown in Table 4-2. 1

393 A regular expression can be matched against a specific field or string by using one
 394 of the two regular expression matching operators, `~` and `!~`. These operators shall
 395 interpret their right-hand operand as a regular expression and their left-hand
 396 operand as a string. If the regular expression matches the string, the `~` expres-
 397 sion shall evaluate to a value of 1, and the `!~` expression shall evaluate to a value
 398 of 0. (The regular expression matching operation is as defined in 2.8.1.2, where a
 399 match occurs on any part of the string unless the regular expression is limited
 400 with the circumflex or dollar-sign special characters.) If the regular expression
 401 does not match the string, the `~` expression shall evaluate to a value of 0, and the
 402 `!~` expression shall evaluate to a value of 1. If the right-hand operand is any
 403 expression other than the lexical token `ERE`, the string value of the expression
 404 shall be interpreted as an extended regular expression, including the escape con-
 405 ventions described above. Note that these same escape conventions also shall be
 406 applied in the determining the value of a string literal (the lexical token `STRING`),
 407 and thus shall be applied a second time when a string literal is used in this con-
 408 text.

409 When an `ERE` token appears as an expression in any context other than as the
 410 right-hand of the `~` or `!~` operator or as one of the built-in function arguments
 411 described below, the value of the resulting expression shall be the equivalent of

412 `$0 ~ /ere/`

413 The `ERE` argument to the `gsub`, `match`, `sub` functions, and the `fs` argument to the
 414 `split` function (see 4.1.7.6.2) shall be interpreted as extended regular expres-
 415 sions. These can be either `ERE` tokens or arbitrary expressions, and shall be inter-
 416 preted in the same manner as the right-hand side of the `~` or `!~` operator.

417 An extended regular expression can be used to separate fields by using the
 418 `-F ERE` option or by assigning a string containing the expression to the built-in
 419 variable `FS`. The default value of the `FS` variable shall be a single `<space>` char-
 420 acter. The following describes `FS` behavior:

(1) If `FS` is a single character:

- (a) If `FS` is `<space>`, skip leading and trailing `<blank>`s; fields shall be delimited by sets of one or more `<blank>`s.
- (b) Otherwise, if `FS` is any other character `c`, fields shall be delimited by each single occurrence of `c`.

(2) Otherwise, the string value of `FS` shall be considered to be an extended regular expression. Each occurrence of a sequence matching the extended regular expression shall delimit fields.

Except in the `gsub`, `match`, `split`, and `sub` built-in functions, regular expression matching shall be based on input records; i.e., record separator characters (the first character of the value of the variable `RS`, `<newline>` by default) cannot be embedded in the expression, and no expression shall match the record separator character. If the record separator is not `<newline>`, `<newline>` characters embedded in the expression can be matched. In those four built-in functions, regular expression matching shall be based on text strings; i.e., any character (including `<newline>` and the record separator) can be embedded in the pattern and an appropriate pattern shall match any character. However, in all `awk` regular expression matching, the use of one or more NUL characters in the pattern, input record, or text string produces undefined results.

4.1.7.5 Patterns

A *pattern* is any valid *expression*, a range specified by two expressions separated by comma, or one of the two special patterns `BEGIN` or `END`.

4.1.7.5.1 Special Patterns

The `awk` utility shall recognize two special patterns, `BEGIN` and `END`. Each `BEGIN` pattern shall be matched once and its associated action executed before the first record of input is read [except possibly by use of the `getline` function (see 4.1.7.6.2) in a prior `BEGIN` action] and before command line assignment is done. Each `END` pattern shall be matched once and its associated action executed after the last record of input has been read. These two patterns shall have associated actions.

`BEGIN` and `END` shall not combine with other patterns. Multiple `BEGIN` and `END` patterns shall be allowed. The actions associated with the `BEGIN` patterns shall be executed in the order specified in the program, as are the `END` actions. An `END` pattern can precede a `BEGIN` pattern in a program.

If an `awk` program consists of only actions with the pattern `BEGIN`, and the `BEGIN` action contains no `getline` function, `awk` shall exit without reading its input when the last statement in the last `BEGIN` action is executed. If an `awk` program consists of only actions with the pattern `END` or only actions with the patterns `BEGIN` and `END`, the input shall be read before the statements in the `END` action(s) are executed.

4.1.7.5.2 Expression Patterns

An expression pattern shall be evaluated as if it were an expression in a Boolean context. If the result is true, the pattern shall be considered to match, and the associated action (if any) shall be executed. If the result is false, the action shall not be executed.

4.1.7.5.3 Pattern Ranges

A pattern range consists of two expressions separated by a comma; in this case, the action shall be performed for all records between a match of the first expression and the following match of the second expression, inclusive. At this point, the pattern range can be repeated starting at input records subsequent to the end of the matched range.

4.1.7.6 Actions

An action is a sequence of statements as shown in the grammar in 4.1.7.7. Any single statement can be replaced by a statement list enclosed in braces. The statements in a statement list shall be separated by <newline>s or semicolons, and shall be executed sequentially in the order that they appear.

The *expression* acting as the conditional in an *if* statement shall be evaluated and if it is nonzero or nonnull, the following *statement* shall be executed; otherwise, if *else* is present, the statement following the *else* shall be executed.

The *if*, *while*, *do ... while*, *for*, *break*, and *continue* statements are based on the C Standard {7} (see 2.9.2), except that the Boolean expressions shall be treated as described in 4.1.7.2, and except in the case of

```
for (variable in array)
```

which shall iterate, assigning each *index* of *array* to *variable* in an unspecified order. The results of adding new elements to *array* within such a *for* loop are undefined. If a *break* or *continue* statement occurs outside of a loop, the behavior is undefined.

The *delete* statement shall remove an individual array element. Thus, the following code shall delete an entire array:

```
for (index in array)
    delete array[index]
```

The *next* statement shall cause all further processing of the current input record to be abandoned. The behavior is undefined if a *next* statement appears or is invoked in a *BEGIN* or *END* action.

The *exit* statement shall invoke all *END* actions in the order in which they occur in the program source and then terminate the program without reading further input. An *exit* statement inside an *END* action shall terminate the program without further execution of *END* actions. If an expression is specified in an *exit* statement, its numeric value shall be the exit status of *awk*, unless subsequent errors are encountered or a subsequent *exit* statement with an expression is

executed.

4.1.7.6.1 Output Statements

Both `print` and `printf` statements shall write to standard output by default. The output shall be written to the location specified by *output_redirection* if one is supplied, as follows:

```
> expression
>> expression
| expression
```

In all cases, the *expression* shall be evaluated to produce a string that is used as a full pathname to write into (for `>` or `>>`) or as a command to be executed (for `|`). Using the first two forms, if the file of that name is not currently open, it shall be opened, creating it if necessary, and using the first form, truncating the file. The output then shall be appended to the file. As long as the file remains open, subsequent calls in which *expression* evaluates to the same string value simply shall append output to the file. The file remains open until the `close` function (see 4.1.7.6.2). is called with an expression that evaluates to the same string value.

The third form shall write output onto a stream piped to the input of a command. The stream shall be created if no stream is currently open with the value of *expression* as its command name. The stream created shall be equivalent to one created by a call to the `popen()` function (see B.3.2) with the value of *expression* as the *command* argument and a value of "w" as the *mode* argument. As long as the stream remains open, subsequent calls in which *expression* evaluates to the same string value shall write output to the existing stream. The stream shall remain open until the `close` function (see 4.1.7.6.2) is called with an expression that evaluates to the same string value. At that time, the stream shall be closed as if by a call to the `pclose()` function (see B.3.2).

As described in detail by the grammar in 4.1.7.7, these output statements shall take a comma-separated list of *expressions* referred in the grammar by the non-terminal symbols `expr_list`, `print_expr_list`, or `print_expr_list_opt`. This list is referred to here as the *expression list*, and each member is referred to as an *expression argument*.

The `print` statement shall write the value of each expression argument onto the indicated output stream separated by the current output field separator (see variable `OFS` above), and terminated by the output record separator (see variable `ORS` above). All expression arguments shall be taken as strings, being converted if necessary; this conversion shall be as described in 4.1.7.2, with the exception that the `printf` format in `OFMT` shall be used instead of the value in `CONVFMT`. An empty expression list shall stand for the whole input record (`$0`). 1 1

The `printf` statement shall produce output based on a notation similar to the File Format Notation used to describe file formats in this standard (see 2.12). Output shall be produced as specified with the first expression argument as the string *<format>* and subsequent expression arguments as the strings *<arg1>* through *<argn>*, with the following exceptions:

- (1) The *format* shall be an actual character string rather than a graphical representation. Therefore, it cannot contain empty character positions. The `<space>` character in the *format* string, in any context other than a *flag* of a conversion specification, shall be treated as an ordinary character that is copied to the output.
- (2) If the character set contains a Δ character and that character appears in the *format* string, it shall be treated as an ordinary character that is copied to the output.
- (3) The *escape sequences* beginning with a backslash character shall be treated as sequences of ordinary characters that are copied to the output. (Note that these same sequences shall be interpreted lexically by `awk` when they appear in literal strings, but they shall not be treated specially by the `printf` statement).
- (4) A *field width* or *precision* can be specified as the `*` character instead of a digit string. In this case the next argument from the expression list shall be fetched and its numeric value taken as the field width or precision.
- (5) The implementation shall not precede or follow output from the `d` or `u` conversion specifications with `<blank>`s not specified by the *format* string.
- (6) The implementation shall not precede output from the `o` conversion specification with leading zeroes not specified by the *format* string.
- (7) For the `c` conversion specification: if the argument has a numeric value, the character whose encoding is that value shall be output. If the value is zero or is not the encoding of any character in the character set, the behavior is undefined. If the argument does not have a numeric value, the first character of the string value shall be output; if the string does not contain any characters the behavior is undefined.
- (8) For each conversion specification that consumes an argument, the next expression argument shall be evaluated. With the exception of the `c` conversion, the value shall be converted (according to the rules specified in 4.1.7.2) to the appropriate type for the conversion specification.
- (9) If there are insufficient expression arguments to satisfy all the conversion specifications in the *format* string, the behavior is undefined.
- (10) If any character sequence in the *format* string begins with a `%` character, but does not form a valid conversion specification, the behavior is unspecified.

Both `print` and `printf` can output at least `{LINE_MAX}` bytes.

4.1.7.6.2 Functions

The `awk` language has a variety of built-in functions: arithmetic, string, input/output, and general.

4.1.7.6.2.1 Arithmetic Functions

The arithmetic functions, except for `int`, shall be based on the C Standard {7}; see 2.9.2. The behavior is undefined in cases where the C Standard {7} specifies that an error be returned or that the behavior is undefined.

<code>atan2(y, x)</code>	Return arctangent of y/x .
<code>cos(x)</code>	Return cosine of x , where x is in radians.
<code>sin(x)</code>	Return sine of x , where x is in radians.
<code>exp(x)</code>	Return the exponential function of x .
<code>log(x)</code>	Return the natural logarithm of x .
<code>sqrt(x)</code>	Return the square root of x .
<code>int(x)</code>	Truncate its argument to an integer. It shall be truncated toward 0 when $x > 0$.
<code>rand()</code>	Return a random number n , such that $0 \leq n < 1$.
<code>srand([expr])</code>	Set the seed value for <code>rand</code> to <i>expr</i> or use the time of day if <i>expr</i> is omitted. The previous seed value shall be returned.

4.1.7.6.2.2 String Functions

The string functions are:

<code>gsub(ere, repl[, in])</code>	Behave like <code>sub</code> (see below), except that it shall replace all occurrences of the regular expression (like the <code>ed</code> utility global substitute) in <code>\$0</code> or in the <i>in</i> argument, when specified.
<code>index(s, t)</code>	Return the position, in characters, numbering from 1, in string <i>s</i> where string <i>t</i> first occurs, or zero if it does not occur at all.
<code>length([s])</code>	Return the length, in characters, of its argument taken as a string, or of the whole record, <code>\$0</code> , if there is no argument.
<code>match(s, ere)</code>	Return the position, in characters, numbering from 1, in string <i>s</i> where the extended regular expression <i>ERE</i> occurs, or zero if it does not occur at all. <code>RSTART</code> shall be set to the starting position (which is the same as the returned value), zero if no match is found; <code>RLENGTH</code> shall be set to the length of the matched string, <code>-1</code> if no match is found.
<code>split(s, a[, fs])</code>	Split the string <i>s</i> into array elements $a[1], a[2], \dots, a[n]$, and returns n . The separation shall be done with the extended regular expression <i>fs</i> or with the field separator

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

623		FS if <i>fs</i> is not given. Each array element shall have a
624		string value when created. If the string assigned to any
625		array element, with any occurrence of the decimal-point
626		character from the current locale changed to a <period>,
627		would be considered a <i>numeric string</i> (see 4.1.7.2), the
628		array element shall also have the numeric value of the
629		<i>numeric string</i> . The effect of a null string as the value of
630		<i>fs</i> is unspecified.
631	<code>sprintf(<i>fmt</i>, <i>expr</i>, ...)</code>	
632		Format the expressions according to the <code>printf</code> format
633		given by <i>fmt</i> and return the resulting string.
634	<code>sub(<i>ere</i>, <i>repl</i>[, <i>in</i>])</code>	
635		Substitute the string <i>repl</i> in place of the first instance of
636		the extended regular expression <i>ERE</i> in string <i>in</i> and
637		return the number of substitutions. An ampersand (&)
638		appearing in the string <i>repl</i> shall be replaced by the string
639		from <i>in</i> that matches the regular expression. An amper-
640		sand preceded by a backslash within <i>repl</i> shall be inter-
641		preted as a literal ampersand character. If <i>in</i> is specified
642		and it is not an <i>lvalue</i> (see 4.1.7.2), the behavior is
643		undefined. If <i>in</i> is omitted, <code>awk</code> shall substitute in the
644		current record (\$0).
645	<code>substr(<i>s</i>, <i>m</i>[, <i>n</i>])</code>	
646		Return the at most <i>n</i> -character substring of <i>s</i> that begins
647		at position <i>m</i> , numbering from 1. If <i>n</i> is missing, the
648		length of the substring shall be limited by the length of
649		the string <i>s</i> .
650	<code>tolower(<i>s</i>)</code>	
651		Return a string based on the string <i>s</i> . Each character in <i>s</i>
652		that is an uppercase letter specified to have a <code>tolower</code>
653		mapping by the <code>LC_CTYPE</code> category of the current locale
654		shall be replaced in the returned string by the lowercase
655		letter specified by the mapping. Other characters in <i>s</i>
		shall be unchanged in the returned string.
656	<code>toupper(<i>s</i>)</code>	
657		Return a string based on the string <i>s</i> . Each character in <i>s</i>
658		that is a lowercase letter specified to have a <code>toupper</code>
659		mapping by the <code>LC_CTYPE</code> category of the current locale
660		shall be replaced in the returned string by the uppercase
661		letter specified by the mapping. Other characters in <i>s</i>
		shall be unchanged in the returned string.
662	All of the preceding functions that take <i>ERE</i> as a parameter expect a pattern or a	
663	string valued expression that is a regular expression as defined in 4.1.7.4.	

4.1.7.6.2.3 Input/Output and General Functions

The input/output and general functions are:

`close(expression)` Close the file or pipe opened by a `print` or `printf` statement or a call to `getline` with the same string-valued *expression*. The limit on the number of open *expression* arguments is implementation defined. If the close was successful, the function shall return zero; otherwise, it shall return nonzero.

`expression | getline [var]`
Read a record of input from a stream piped from the output of a command. The stream shall be created if no stream is currently open with the value of *expression* as its command name. The stream created shall be equivalent to one created by a call to the `popen()` function with the value of *expression* as the *command* argument and a value of "r" as the *mode* argument. As long as the stream remains open, subsequent calls in which *expression* evaluates to the same string value shall read subsequent records from the file. The stream shall remain open until the `close` function is called with an *expression* that evaluates to the same string value. At that time, the stream shall be closed as if by a call to the `pclose()` function. If *var* is missing, \$0 and NF shall be set; otherwise, *var* shall be set.

`getline`
Set \$0 to the next input record from the current input file. This form of `getline` shall set the NF, NR, and FNR variables.

`getline var`
Set variable *var* to the next input record from the current input file. This form of `getline` shall set the FNR and NR variables.

`getline [var] < expression`
Read the next record of input from a named file. The *expression* shall be evaluated to produce a string that is used as a full pathname. If the file of that name is not currently open, it shall be opened. As long as the stream remains open, subsequent calls in which *expression* evaluates to the same string value shall read subsequent records from the file. The file shall remain open until the `close` function is called with an *expression* that evaluates to the same string value. If *var* is missing, \$0 and NF shall be set; otherwise, *var* shall be set.

`system(expression)`
Execute the command given by *expression* in a manner equivalent to the `system()` function [see B.3.1] and return

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

the exit status of the command.

All forms of `getline` shall return 1 for successful input, zero for end of file, and -1 for an error.

4.1.7.6.2.4 User-Defined Functions

The `awk` language also shall provide user-defined functions. Such functions can be defined as:

```
function name(args,...) { statements }
```

A function can be referred to anywhere in an `awk` program; in particular, its use can precede its definition. The scope of a function shall be global.

Function arguments can be either scalars or arrays; the behavior is undefined if an array name is passed as an argument that the function uses as a scalar, or if a scalar expression is passed as an argument that the function uses as an array. Function arguments shall be passed by value if scalar and by reference if array name. Argument names shall be local to the function; all other variable names shall be global. The same name shall not be used as both an argument name and as the name of a function or a special `awk` variable. The same name shall not be used both as a variable name with global scope and as the name of a function. The same name shall not be used within the same scope both as a scalar variable and as an array.

The number of parameters in the function definition need not match the number of parameters in the function call. Excess formal parameters can be used as local variables. If fewer arguments are supplied in a function call than are in the function definition, the extra parameters that are used in the function body as scalars shall be initialized with a string value of the null string and a numeric value of zero, and the extra parameters that are used in the function body as arrays shall be initialized as empty arrays. If more arguments are supplied in a function call than are in the function definition, the behavior is undefined.

When invoking a function, no white space can be placed between the function name and the opening parenthesis. The implementation shall permit function calls to be nested, and for recursive calls to be made upon functions. Upon return from any nested or recursive function call, the values of all of the calling function's parameters shall be unchanged, except for array parameters passed by reference. The `return` statement can be used to return a value. If a `return` statement appears outside of a function definition, the behavior is undefined.

In the function definition, `<newline>`s shall be optional before the opening brace and after the closing brace. Function definitions can appear anywhere in the program where a *pattern-action* pair is allowed.

4.1.7.7 `awk` Grammar

The grammar in this subclause and the lexical conventions in the following subclause shall together describe the syntax for `awk` programs. The general conventions for this style of grammar are described in 2.1.2. A valid program can be

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

749 represented as the nonterminal symbol *program* in the grammar. Any discrepan-
 750 cies found between this grammar and other descriptions in this clause shall be
 751 resolved in favor of this grammar.

```

752 %token  NAME NUMBER STRING ERE NEWLINE
753 %token  FUNC_NAME      /* name followed by '(' without white space */
754
755 /* Keywords */
756 %token  Begin  End
757 /*      'BEGIN' 'END' */
758
759 %token  Break  Continue  Delete  Do  Else
760 /*      'break' 'continue' 'delete' 'do' 'else' */
761
762 %token  Exit  For  Function  If  In
763 /*      'exit' 'for' 'function' 'if' 'in' */
764
765 %token  Next  Print  Printf  Return  While
766 /*      'next' 'print' 'printf' 'return' 'while' */
767
768 /* Reserved function names */
769 %token  BUILTIN_FUNC_NAME /* one token for the following:
770 * atan2 cos sin exp log sqrt int rand srand
771 * gsub index length match split sprintf sub substr
772 * tolower toupper close system
773 */
774 %token  GETLINE /* Syntactically different from other built-ins */
775
776 /* Two-character tokens */
777 %token  ADD_ASSIGN SUB_ASSIGN MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN POW_ASSIGN
778 /*      '+='      '-='      '*='      '/='      '%='      '^=' */
779
780 %token  OR  AND  NO_MATCH EQ  LE  GE  NE  INCR  DECR  APPEND
781 /*      '||' '&&' '!~'      '==' '<=' '>=' '!=' '++' '--' '>>' */
782
783 /* One-character tokens */
784 %token  '{' '}' '(' ')' '[' ']' ',' ';'
785 %token  '+' '-' '*' '%' '^' '!' '>' '<' '|' '?' ':' '~' '$' '='
786
787 %start  program
788 %%
789
790 program:
791     item_list
792     | actionless_item_list
793     ;
794
795 item_list:
796     newline_opt
797     | actionless_item_list item terminator
798     | item_list item terminator
799     | item_list action terminator
800     ;
801
802 actionless_item_list:
803     item_list pattern terminator
804     | actionless_item_list pattern terminator
805     ;
806
807 item:
808     pattern action
  
```

```

796         | Function NAME '(' param_list_opt ')' newline_opt action
797         | Function FUNC_NAME '(' param_list_opt ')' newline_opt action
798         ;

799 param_list_opt:
800     /* empty */
801     | param_list
802     ;

803 param_list:
804     NAME
805     | param_list ',' NAME
806     ;

807 pattern:
808     Begin
809     | End
810     | expr
811     | expr ',' newline_opt expr
812     ;

813 action:
814     '{' newline_opt '}'
815     | '{' newline_opt terminated_statement_list '}'
816     | '{' newline_opt unterminated_statement_list '}'
817     ;

818 terminator:
819     ';'
820     | NEWLINE
821     | terminator NEWLINE ';'
822     ;

823 terminated_statement_list:
824     terminated_statement
825     | terminated_statement_list terminated_statement
826     ;

827 unterminated_statement_list:
828     unterminated_statement
829     | terminated_statement_list unterminated_statement
830     ;

831 terminated_statement:
832     action newline_opt
833     | If '(' expr ')' newline_opt terminated_statement
834     | Else newline_opt terminated_statement
835     | While '(' expr ')' newline_opt terminated_statement
836     | For '(' simple_statement_opt ';' expr_opt ';' simple_statement_opt ')'
837     | newline_opt terminated_statement
838     | For '(' NAME In NAME ')' newline_opt terminated_statement
839     | ';' newline_opt
840     | terminatable_statement NEWLINE newline_opt
841     | terminatable_statement ';' newline_opt
842     ;

843 unterminated_statement:
844     terminatable_statement
845     | If '(' expr ')' newline_opt unterminated_statement
846     | If '(' expr ')' newline_opt terminated_statement
847     | Else newline_opt unterminated_statement

```

2

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```

848         | While '(' expr ')' newline_opt unterminated_statement
849         | For '(' simple_statement_opt ';' expr_opt ';' simple_statement_opt ')'
850           newline_opt unterminated_statement
851         | For '(' NAME In NAME ')' newline_opt unterminated_statement
852         ;

853     terminatable_statement:
854         simple_statement
855         | Break
856         | Continue
857         | Next
858         | Exit expr_opt
859         | Return expr_opt
860         | Do newline_opt terminated_statement While '(' expr ')'
861         ;

862     simple_statement_opt:
863         /* empty */
864         | simple_statement
865         ;

866     simple_statement:
867         Delete NAME '[' expr_list ']'
868         | expr
869         | print_statement
870         ;

871     print_statement:
872         simple_print_statement
873         | simple_print_statement output_redirection
874         ;

875     simple_print_statement:
876         Print print_expr_list_opt
877         | Print '(' multiple_expr_list ')'
878         | Printf print_expr_list
879         | Printf '(' multiple_expr_list ')'
880         ;

881     output_redirection:
882         '>' expr
883         | APPEND expr
884         | '|' expr
885         ;

886     expr_list_opt:
887         /* empty */
888         | expr_list
889         ;

890     expr_list:
891         expr
892         | multiple_expr_list
893         ;

894     multiple_expr_list:
895         expr ',' newline_opt expr
896         | multiple_expr_list ',' newline_opt expr
897         ;

898     expr_opt:
899         /* empty */

```

```

900         | expr
901         ;
902     expr:
903         unary_expr
904         | non_unary_expr
905         ;
906     unary_expr:
907         '+' expr
908         | '-' expr
909         | unary_expr '^' expr
910         | unary_expr '*' expr
911         | unary_expr '/' expr
912         | unary_expr '%' expr
913         | unary_expr '+' expr
914         | unary_expr '-' expr
915         | unary_expr non_unary_expr
916         | unary_expr '<' expr
917         | unary_expr LE expr
918         | unary_expr NE expr
919         | unary_expr EQ expr
920         | unary_expr '>' expr
921         | unary_expr GE expr
922         | unary_expr '~' expr
923         | unary_expr NO_MATCH expr
924         | unary_expr In NAME
925         | unary_expr AND newline_opt expr
926         | unary_expr OR newline_opt expr
927         | unary_expr '?' expr ':' expr
928         | unary_input_function
929         ;
930     non_unary_expr:
931         '(' expr ')'
932         | '!' expr
933         | non_unary_expr '^' expr
934         | non_unary_expr '*' expr
935         | non_unary_expr '/' expr
936         | non_unary_expr '%' expr
937         | non_unary_expr '+' expr
938         | non_unary_expr '-' expr
939         | non_unary_expr non_unary_expr
940         | non_unary_expr '<' expr
941         | non_unary_expr LE expr
942         | non_unary_expr NE expr
943         | non_unary_expr EQ expr
944         | non_unary_expr '>' expr
945         | non_unary_expr GE expr
946         | non_unary_expr '~' expr
947         | non_unary_expr NO_MATCH expr
948         | non_unary_expr In NAME
949         | '(' multiple_expr_list ')' In NAME
950         | non_unary_expr AND newline_opt expr
951         | non_unary_expr OR newline_opt expr
952         | non_unary_expr '?' expr ':' expr
953         | NUMBER
954         | STRING

```

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

```

955         | lvalue
956         | ERE
957         | lvalue INCR
958         | lvalue DECR
959         | INCR lvalue
960         | DECR lvalue
961         | lvalue POW_ASSIGN expr
962         | lvalue MOD_ASSIGN expr
963         | lvalue MUL_ASSIGN expr
964         | lvalue DIV_ASSIGN expr
965         | lvalue ADD_ASSIGN expr
966         | lvalue SUB_ASSIGN expr
967         | lvalue '=' expr
968         | FUNC_NAME '(' expr_list_opt ')' /* no white space allowed */
969         | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
970         | BUILTIN_FUNC_NAME
971         | non_unary_input_function
972         ;

973 print_expr_list_opt:
974     /* empty */
975     | print_expr_list
976     ;

977 print_expr_list:
978     print_expr
979     | print_expr_list ',' newline_opt print_expr
980     ;

981 print_expr:
982     unary_print_expr
983     | non_unary_print_expr
984     ;

985 unary_print_expr:
986     '+' print_expr
987     | '-' print_expr
988     | unary_print_expr '^' print_expr
989     | unary_print_expr '*' print_expr
990     | unary_print_expr '/' print_expr
991     | unary_print_expr '%' print_expr
992     | unary_print_expr '+' print_expr
993     | unary_print_expr '-' print_expr
994     | unary_print_expr non_unary_print_expr
995     | unary_print_expr '~' print_expr
996     | unary_print_expr NO_MATCH print_expr
997     | unary_print_expr In NAME
998     | unary_print_expr AND newline_opt print_expr
999     | unary_print_expr OR newline_opt print_expr
1000    | unary_print_expr '?' print_expr ':' print_expr
1001    ;

1002 non_unary_print_expr:
1003    '(' expr ')'
1004    | '!' print_expr
1005    | non_unary_print_expr '^' print_expr
1006    | non_unary_print_expr '*' print_expr
1007    | non_unary_print_expr '/' print_expr
1008    | non_unary_print_expr '%' print_expr

```

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

```

1009     | non_unary_print_expr '+' print_expr
1010     | non_unary_print_expr '-' print_expr
1011     | non_unary_print_expr non_unary_print_expr
1012     | non_unary_print_expr '~' print_expr
1013     | non_unary_print_expr NO_MATCH print_expr
1014     | non_unary_print_expr In NAME
1015     | '(' multiple_expr_list ')' In NAME
1016     | non_unary_print_expr AND newline_opt print_expr
1017     | non_unary_print_expr OR newline_opt print_expr
1018     | non_unary_print_expr '?' print_expr ':' print_expr
1019     | NUMBER
1020     | STRING
1021     | lvalue
1022     | ERE
1023     | lvalue INCR
1024     | lvalue DECR
1025     | INCR lvalue
1026     | DECR lvalue
1027     | lvalue POW_ASSIGN print_expr
1028     | lvalue MOD_ASSIGN print_expr
1029     | lvalue MUL_ASSIGN print_expr
1030     | lvalue DIV_ASSIGN print_expr
1031     | lvalue ADD_ASSIGN print_expr
1032     | lvalue SUB_ASSIGN print_expr
1033     | lvalue '=' print_expr
1034     | FUNC_NAME '(' expr_list_opt ')' /* no white space allowed */
1035     | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
1036     | BUILTIN_FUNC_NAME
1037     ;

1038 lvalue:
1039     NAME
1040     | NAME '[' expr_list ']'
1041     | '$' expr
1042     ;

1043 non_unary_input_function:
1044     simple_get
1045     | simple_get '<' expr
1046     | non_unary_expr '|' simple_get
1047     ;

1048 unary_input_function:
1049     unary_expr '|' simple_get
1050     ;

1051 simple_get:
1052     GETLINE
1053     | GETLINE lvalue
1054     ;

1055 newline_opt:
1056     /* empty */
1057     | newline_opt NEWLINE
1058     ;

```

1059 This grammar has several ambiguities that shall be resolved as follows:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

- Operator precedence and associativity shall be as described in Table 4-1.
- In case of ambiguity, an `else` shall be associated with the most immediately preceding `if` that would satisfy the grammar.

4.1.7.8 `awk` Lexical Conventions

The lexical conventions for `awk` programs, with respect to the preceding grammar, shall be as follows:

- (1) Except as noted, `awk` shall recognize the longest possible token or delimiter beginning at a given point.
- (2) A comment shall consist of any characters beginning with the number sign character and terminated by, but excluding the next occurrence of, a `<newline>` character. Comments shall have no effect, except to delimit lexical tokens.
- (3) The character `<newline>` shall be recognized as the token `NEWLINE`.
- (4) A backslash character immediately followed by a `<newline>` character shall have no effect.
- (5) The token `STRING` shall represent a string constant. A string constant shall begin with the character `"`. Within a string constant, a backslash character shall be considered to begin an escape sequence as specified in Table 2-15 (see 2.12). In addition, the escape sequences in Table 4-2 shall be recognized. A `<newline>` character shall not occur within a string constant. A string constant shall be terminated by the first unescaped occurrence of the character `"` after the one that begins the string constant. The value of the string shall be the sequence of all unescaped characters and values of escape sequences between, but not including, the two delimiting `"` characters.
- (6) The token `ERE` represents an extended regular expression constant. An `ERE` constant shall begin with the slash character. Within an `ERE` constant, a `<backslash>` character shall be considered to begin an escape sequence as specified in Table 2-15 (see 2.12). In addition, the escape sequences in Table 4-2 shall be recognized. A `<newline>` character shall not occur within an `ERE` constant. An `ERE` constant shall be terminated by the first unescaped occurrence of the slash character after the one that begins the string constant. The extended regular expression represented by the `ERE` constant shall be the sequence of all unescaped characters and values of escape sequences between, but not including, the two delimiting slash characters.
- (7) A `<blank>` shall have no effect, except to delimit lexical tokens or within `STRING` or `ERE` tokens.
- (8) The token `NUMBER` shall represent a numeric constant. Its form and numeric value shall be equivalent to the either of the tokens `floating-constant` or `integer-constant` as specified by the C Standard [7], with the following exceptions:

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

Table 4-2 – awk Escape Sequences

Escape Sequence	Description	Meaning	
\ "	<backslash> <quotation-mark>	<quotation-mark> character	
\ /	<backslash> <slash>	<slash> character	
\ddd	<backslash> followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0, (i.e., representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the one-, two-, or three-digit octal integer. If the size of a byte on the system is greater than nine bits, the valid escape sequence used to represent a byte is implementation defined. Multibyte characters require multiple, concatenated escape sequences of this type, including the leading \ for each byte.	1 1 1 1 1 1 1 1 1 1
\c	<backslash> followed by any character not described in this table or in Table 2-15	Undefined	

(a) An integer constant cannot begin with 0x or include the hexadecimal digits a, b, c, d, e, f, A, B, C, D, E, or F.

(b) The value of an integer constant beginning with 0 shall be taken in decimal rather than octal.

(c) An integer constant cannot include a suffix (u, U, l, or L).

(d) A floating constant cannot include a suffix (f, F, l, or L).

If the value is too large or too small to be representable (see 2.9.2.1), the behavior is undefined.

(9) A sequence of underscores, digits, and alphabetic characters from the portable character set (see 2.4), beginning with an underscore or alphabetic, shall be considered a word.

(10) The following words are keywords that shall be recognized as individual tokens; the name of the token is the same as the keyword:

BEGIN	delete	for	in	printf
END	do	function	next	return
break	else	getline	print	while
continue	exit	if		

(11) The following words are names of built-in functions and shall be recognized as the token BUILTIN_FUNC_NAME:

atan2	index	match	sprintf	substr
close	int	rand	sqrt	system

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1142	cos	length	sin	srand	tolower
1143	exp	log	split	sub	toupper
1144	gsub				

1145 The above-listed keywords and names of built-in functions are considered
1146 reserved words.

1147 (12) The token `NAME` shall consist of a word that is not a keyword or a name of
1148 a built-in function and is not followed immediately (without any delimiters)
1149 by the `(` character.

1150 (13) The token `FUNC_NAME` shall consist of a word that is not a keyword or a
1151 name of a built-in function, followed immediately (without any delimiters)
1152 by the `(` character. The `(` character shall not be included as part of
1153 the token.

1154 (14) The following two-character sequences shall be recognized as the named
1155 tokens:

	<u>Token Name</u>	<u>Sequence</u>	<u>Token Name</u>	<u>Sequence</u>
1156	ADD_ASSIGN	+=	NO_MATCH	!~
1157	SUB_ASSIGN	-=	EQ	==
1158	MUL_ASSIGN	*=	LE	<=
1159	DIV_ASSIGN	/=	GE	>=
1160	MOD_ASSIGN	%=	NE	!=
1161	POW_ASSIGN	=	INCR	++
1162	OR		DECR	--
1163	AND	&&	APPEND	>>
1164				

1165 (15) The following single characters shall be recognized as tokens whose
1166 names are the character:

1167 <newline> { } () [] , ; + - * % ^ ! > < | ? : ~ \$ =

1168 There is a lexical ambiguity between the token `ERE` and the tokens `/` and
1169 `DIV_ASSIGN`. When an input sequence begins with a slash character in any syntactic
1170 context where the token `/` or `DIV_ASSIGN` could appear as the next token in
1171 a valid program, the longer of those two tokens that can be recognized shall be
1172 recognized. In any other syntactic context where the token `ERE` could appear as
1173 the next token in a valid program, the token `ERE` shall be recognized.

1174 4.1.8 Exit Status

1175 The `awk` utility shall exit with one of the following values:

1176	0	All input files were processed successfully.
1177	>0	An error occurred.

1178 The exit status can be altered within the program by using an `exit` expression.

4.1.9 Consequences of Errors

If any *file* operand is specified and the named file cannot be accessed, *awk* shall write a diagnostic message to standard error and terminate without any further action.

If the program specified by either the *program* operand or the *progfile* operand(s) is not a valid *awk* program (as specified in 4.1.7), the behavior is undefined.

4.1.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The *awk* program specified in the command line is most easily specified within single-quotes (e.g., '*program*') for applications using *sh*, because *awk* programs commonly contain characters that are special to the shell, including double-quotes. In the cases where an *awk* program contains single-quote characters, it is usually easiest to specify most of the program as strings within single-quotes concatenated by the shell with quoted single-quote characters. For example,

```
awk '/'\''/' { print "quote:", $0 }
```

prints all lines from the standard input containing a single-quote character, prefixed with *quote:*.

The following are examples of simple *awk* programs:

- (1) Write to the standard output all input lines for which field 3 is greater than 5.

```
$3 > 5
```

- (2) Write every tenth line.

```
(NR % 10) == 0
```

- (3) Write any line with a substring matching the regular expression.

```
/(G|D)(2[0-9][[:alpha:]]*)/
```

- (4) Write any line in which the second field matches the regular expression and the fourth field does not.

```
$2 ~ /xyz/ && $4 !~ /xyz/
```

- (5) Write any line in which the second field contains a backslash.

```
$2 ~ /\\"/>

```

- (6) Write any line in which the second field contains a backslash. Note that backslash escapes are interpreted twice, once in lexical processing of the string and once in processing the regular expression.

```
$2 ~ "\\\"/>

```

- (7) Write the second to the last and the last field in each line. Separate the fields by a colon.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

```

1215         {OFS=":";print $(NF-1), $NF}
1216     (8) Write the line number and number of fields in each line. The three
1217         strings representing the line number, the colon and the number of fields
1218         are concatenated and that string is written to standard output.
1219         {print NR ":" NF}
1220     (9) Write lines longer than 72 characters.
1221         {length($0) > 72}
1222     (10) Write first two fields in opposite order separated by the OFS:
1223         { print $2, $1 }
1224     (11) Same, with input fields separated by comma and/or <space>s and
1225         <tab>s:
1226         BEGIN { FS = ",[ \t]*|[ \t]+" }
1227         { print $2, $1 }
1228     (12) Add up first column, print sum and average.
1229         {s += $1 }
1230         END {print "sum is ", s, " average is", s/NR}
1231     (13) Write fields in reverse order, one per line (many lines out for each line
1232         in):
1233         { for (i = NF; i > 0; --i) print $i }
1234     (14) Write all lines between occurrences of the strings start and stop:
1235         /start/, /stop/
1236     (15) Write all lines whose first field is different from the previous one:
1237         $1 != prev { print; prev = $1 }
1238     (16) Simulate echo:
1239         BEGIN {
1240             for (i = 1; i < ARGV; ++i)
1241                 printf "%s%s", ARGV[i], i==ARGC-1?"\n":""
1242         }
1243     (17) Write the path prefixes contained in the PATH environment variable, one
1244         per line:
1245         BEGIN {
1246             n = split (ENVIRON["PATH"], path, ":")
1247             for (i = 1; i <= n; ++i)
1248                 print path[i]
1249         }
1250     (18) If there is a file named “input” containing page headers of the form:
1251         Page #
1252         and a file named “program” that contains:

```

```

1253         /Page/{ $2 = n++; }
1254         { print }

```

then the command line:

```

1256     awk -f program n=5 input

```

will print the file “input,” filling in page numbers starting at 5.

The index, length, match, and substr should not be confused with similar functions in the C Standard {7}; the awk versions deal with characters, while the C Standard {7} deals with bytes.

To forestall any possible confusion, where strings are used as the name of a file or pipeline, the strings must be textually identical. The terminology “same string value” implies that “equivalent strings,” even those that differ only by <space>s, represent different files.

History of Decisions Made

This description is based on the new awk, “nawk,” (see *The AWK Programming Language* {B21}), which introduced a number of new features to the historical awk:

- (1) New keywords: delete, do, function, return
- (2) New built-in functions: atan2, cos, sin, rand, srand, gsub, sub, match, close, system
- (3) New predefined variables: FNR, ARGV, RSTART, RLENGTH, SUBSEP
- (4) New expression operators: ?:, ^
- (5) The FS variable and the third argument to split are now treated as extended regular expressions.
- (6) The operator precedence has changed to more closely match C. Two examples of code that operate differently are:

```

1278         while ( n /= 10 > 1) ...
1279         if (!"wk" ~ /bwk/) ...

```

Several features have been added based on newer implementations of awk:

- (1) Multiple instances of -f *progfile* are permitted.
- (2) New option: -v *assignment*
- (3) New predefined variable: ENVIRON
- (4) New built-in functions: toupper, tolower
- (5) More formatting capabilities added to printf to match the C Standard {7}.

Regular expressions have been extended somewhat from traditional implementations to make them a pure superset of Extended Regular Expressions as defined by this standard (see 2.8.4). The main extensions are internationalization

features and interval expressions. Traditional implementations of `awk` have long supported `<backslash>` escape sequences as an extension to regular expressions, and this extension has been retained despite inconsistency with other utilities. The number of escape sequences recognized in both regular expressions and strings has varied (generally increasing with time) among implementations. The set specified by the standard includes most sequences known to be supported by popular implementations and by the C Standard [7]. One sequence that is not supported is hexadecimal value escapes beginning with `"\x"`. This would allow values expressed in more than 9 bits to be used within `awk` as in the C Standard [7]. However, because this syntax has a nondeterministic length, it does not permit the subsequent character to be a hexadecimal digit. This limitation can be worked around in the C language by the use of lexical string concatenation. In the `awk` language, concatenation could also be a solution for strings, but not for regular expressions (either lexical `ERE` tokens or strings used dynamically as regular expressions). Because of this limitation, the feature has not been added to POSIX.2.

When a string variable is used in a context where an `ERE` normally appears (where the lexical token `ERE` is used in the grammar) the string does not contain the literal slashes.

Some versions of `awk` allow the form:

```
func name(args,...) { statements }
```

This has been deprecated by the language's authors, who have asked that it not be included in the standard.

Traditional implementations of `awk` produce an error if a `next` statement is executed in a `BEGIN` action, and cause `awk` to terminate if a `next` statement is executed in an `END` action. This behavior has not been documented, and it was not believed that it was necessary to standardize it.

The specification of conversions between string and numeric values is much more detailed than in the documentation of traditional implementations or in *The AWK Programming Language* [B21]. Although most of the behavior is designed to be intuitive, the details are necessary to ensure compatible behavior from different implementations. This is especially important in relational expressions, since the types of the operands determine whether a string or numeric comparison is performed. From the perspective of an application writer, it is usually sufficient to expect intuitive behavior and to force conversions (by adding zero or concatenating a null string) when the type of an expression does not obviously match what is needed. The intent has been to specify existing practice in almost all cases. The one exception is that, in traditional implementations, variables and constants maintain both string and numeric values after their original value is converted by any use. This means that referencing a variable or constant can have unexpected side effects. For example, with traditional implementations the following program:

```

1332     {
1333         a = "+2"
1334         b = 2
1335         if (NR % 2)
1336             c = a + b
1337         if (a == b)
1338             print "numeric comparison"
1339         else
1340             print "string comparison"
1341     }

```

1342 would perform a numeric comparison (and output **numeric comparison**) for
 1343 each odd-numbered line, but perform a string comparison (and output **string**
 1344 **comparison**) for each even-numbered line. POSIX.2 ensures that comparisons 1
 1345 will be numeric if necessary. With traditional implementations, the following pro- 1
 1346 gram:

```

1347     BEGIN {
1348         OFMT = "%e"
1349         print 3.14
1350         OFMT = "%f"
1351         print 3.14
1352     }

```

1353 would output **3.140000e+00** twice, because in the second `print` statement the
 1354 constant `3.14` would have a string value from the previous conversion. The stan-
 1355 dard requires that the output of the second `print` statement be **3.140000**. The
 1356 behavior of traditional implementations was seen as too unintuitive and
 1357 unpredictable.

1358 However, a further modification was made in Draft 11. It was pointed out that
 1359 with the Draft 10 rules, the following script would print nothing:

```

1360     BEGIN {
1361         y[1.5] = 1
1362         OFMT = "%e"
1363         print y[1.5]
1364     }

```

1365 Therefore, a new variable, `CONVFMT`, was introduced. The `OFMT` variable is now
 1366 restricted to affecting output conversions of numbers to strings and `CONVFMT`
 1367 is used for internal conversions, such as comparisons or array indexing. The default
 1368 value is the same as that for `OFMT`, so unless a program changes `CONVFMT` (which
 1369 no historical program would do), it will receive the historical behavior associated
 1370 with internal string conversions.

1371 The POSIX `awk` lexical and syntactic conventions are specified more formally than
 1372 in other sources. Again the intent has been to specify existing practice. One con-
 1373 vention that may not be obvious from the formal grammar as in other verbal
 1374 descriptions is where `<newline>s` are acceptable. There are several obvious
 1375 placements such as terminating a statement, and a backslash can be used to
 1376 escape `<newline>s` between any lexical tokens. In addition, `<newline>s`
 1377 without backslashes can follow a comma, an open brace, logical AND operator

(`&&`), logical OR operator (`||`), the `do` keyword, the `else` keyword, and the closing parenthesis of an `if`, `for`, or `while` statement. For example:

```
{ print $1,
    $2 }
```

The requirement that `awk` add a trailing `<newline>` to the *program* argument text is to simplify the grammar, making it match a text file in form. There is no way for an application or test suite to determine whether a literal `<newline>` is added or whether `awk` simply acts as if it did.

Because the concatenation operation is represented by adjacent expressions rather than an explicit operator, it is often necessary to use parentheses to enforce the proper evaluation precedence.

The overall `awk` syntax has always been based on the C language, with a few features from the shell command language and other sources. Because of this, it is not completely compatible with any other language, which has caused confusion for some users. It is not the intent of this standard to address such issues. The standard has made a few relatively minor changes toward making the language more compatible with the C language as specified by the C Standard [7]; most of these changes are based on similar changes in recent implementations, as described above. There remain several C language conventions that are not in *awk*. One of the notable ones is the comma operator, which is commonly used to specify multiple expressions in the C language `for` statement. Also, there are various places where `awk` is more restrictive than the C language regarding the type of expression that can be used in a given context. These limitations are due to the different features that the `awk` language does provide.

This standard requires several changes from traditional implementations in order to support internationalization. Probably the most subtle of these is the use of the decimal-point character, defined by the `LC_NUMERIC` category of the locale, in representations of floating point numbers. This locale-specific character is used in recognizing numeric input, in converting between strings and numeric values, and in formatting output. However, regardless of locale, the period character (the decimal-point character of the POSIX Locale) is the decimal-point character recognized in processing `awk` programs (including assignments in command-line arguments). This is essentially the same convention as the one used in the C Standard [7]. The difference is that the C language includes the `setlocale()` function, which permits an application to modify its locale. Because of this capability, a C application begins executing with its locale set to the C locale, and only executes in the environment-specified locale after an explicit call to `setlocale()`. However, adding such an elaborate new feature to the `awk` language was seen as inappropriate for POSIX.2. It is possible to explicitly execute an `awk` program in any desired locale by setting the environment in the shell.

The behavior in the case of invalid `awk` programs (including lexical, syntactic, and semantic errors) is undefined because it was considered overly limiting on implementations to specify. In most cases such errors can be expected to produce a diagnostic and a nonzero exit status. However, some implementations may choose to extend the language in ways that make use of certain invalid constructs.

Other invalid constructs might be deemed worthy of a warning but otherwise cause some reasonable behavior. Still other constructs may be very difficult to detect in some implementations. Also, different implementations might detect a given error during an initial parsing of the program (before reading any input files) while others might detect it when executing the program after reading some input. Implementors should be aware that diagnosing errors as early as possible and producing useful diagnostics can ease debugging of applications, and thus make an implementation more usable.

The unspecified behavior from using multicharacter RS values is to allow possible future extensions based on regular expressions used for record separators. Historical implementations take the first character of the string and ignore the others.

The undefined behavior resulting from NULs in regular expressions allows future extensions for the GNU `gawk` program to process binary data.

Unspecified behavior when `split(string,array,<null>)` is used is to allow a proposed future extension that would split up a string into an array of individual characters.

4.2 `basename` — Return nondirectory portion of pathname

4.2.1 Synopsis

`basename string [suffix]`

4.2.2 Description

The *string* operand shall be treated as a pathname, as defined in 2.2.2.102. The string *string* shall be converted to the filename corresponding to the last pathname component in *string* and then the suffix string *suffix*, if present, shall be removed. This shall be done by performing actions equivalent to the following steps in order:

- (1) If *string* is `//`, it is implementation defined whether steps (2) through (5) are skipped or processed.
- (2) If *string* consists entirely of slash characters, *string* shall be set to a single slash character. In this case, skip steps (3) through (5).
- (3) If there are any trailing slash characters in *string*, they shall be removed.
- (4) If there are any slash characters remaining in *string*, the prefix of *string* up to and including the last slash character in *string* shall be removed.
- (5) If the *suffix* operand is present, is not identical to the characters remaining in *string*, and is identical to a suffix of the characters remaining in *string*, the suffix *suffix* shall be removed from *string*. Otherwise, *string*

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1459 shall not be modified by this step. It shall not be considered an error if
 1460 *suffix* is not found in *string*.

1461 The resulting string shall be written to standard output.

1462 **4.2.3 Options**

1463 None.

1464 **4.2.4 Operands**

1465 The following operands shall be supported by the implementation:

1466 *string* A string.

1467 *suffix* A string.

1468 **4.2.5 External Influences**

1469 **4.2.5.1 Standard Input**

1470 None.

1471 **4.2.5.2 Input Files**

1472 None.

1473 **4.2.5.3 Environment Variables**

1474 The following environment variables shall affect the execution of *basename*:

1475 **LANG** This variable shall determine the locale to use for the
 1476 locale categories when both **LC_ALL** and the correspond-
 1477 ing environment variable (beginning with **LC_**) do not
 1478 specify a locale. See 2.6.

1479 **LC_ALL** This variable shall determine the locale to be used to over-
 1480 ride any values for locale categories specified by the set-
 1481 tings of **LANG** or any environment variables beginning
 1482 with **LC_**.

1483 **LC_CTYPE** This variable shall determine the locale for the interpreta-
 1484 tion of sequences of bytes of text data as characters (e.g.,
 1485 single- versus multibyte characters in arguments).

1486 **LC_MESSAGES** This variable shall determine the language in which mes-
 1487 sages should be written.

1488 **4.2.5.4 Asynchronous Events**

1489 Default.

1490 **4.2.6 External Effects**

1491 **4.2.6.1 Standard Output**

1492 The `basename` utility shall write a line to the standard output in the following
1493 format:

1494 "%s\n", *<resulting string>*

1495 **4.2.6.2 Standard Error**

1496 Used only for diagnostic messages.

1497 **4.2.6.3 Output Files**

1498 None.

1499 **4.2.7 Extended Description**

1500 None.

1501 **4.2.8 Exit Status**

1502 The `basename` utility shall exit with one of the following values:

1503 0 Successful completion.

1504 >0 An error occurred.

1505 **4.2.9 Consequences of Errors**

1506 Default.

1507 **4.2.10 Rationale.** *(This subclause is not a part of P1003.2)*

1508 **Examples, Usage**

1509 If the string *string* is a valid pathname,

1510 `$(basename "string")`

1511 produces a filename that could be used to open the file named by *string* in the
1512 directory returned by

```
1513      $(dirname "string")
```

1514 If the string *string* is not a valid pathname, the same algorithm is used, but the
 1515 result need not be a valid filename. The `basename` utility is not expected to make
 1516 any judgements about the validity of *string* as a pathname; it just follows the
 1517 specified algorithm to produce a result string.

1518 The following shell script compiles `/usr/src/cmd/cat.c` and moves the output
 1519 to a file named `cat` in the current directory when invoked with the argument
 1520 `/usr/src/cmd/cat` or with the argument `/usr/src/cmd/cat.c`:

```
1521      c89 $(dirname "$1")/$(basename "$1" .c).c
1522      mv a.out $(basename "$1" .c)
```

1523 History of Decisions Made

1524 The POSIX.1 {8} definition of *pathname* allows trailing slashes on a *pathname*
 1525 naming a directory. Some historical implementations have not allowed trailing
 1526 slashes and thus treated *pathnames* of this form in other ways. Existing imple-
 1527 mentations also differ in their handling of *suffix* when *suffix* matches the entire
 1528 string left after removing the directory part of *string*.

1529 The behaviors of `basename` and `dirname` in this standard have been coordinated
 1530 so that when *string* is a valid *pathname*

```
1531      $(basename "string")
```

1532 would be a valid filename for the file in the directory

```
1533      $(dirname "string")
```

1534 This would not work for the versions of these utilities in earlier drafts due to the
 1535 way it specified handling of trailing slashes.

1536 Since the definition of *pathname* in 2.2.2.102 specifies implementation-defined
 1537 behavior for *pathnames* starting with two slash characters, Draft 11 has been
 1538 changed to specify similar implementation-defined behavior for the `basename`
 1539 and `dirname` utilities. On implementations where the *pathname* `//` is always
 1540 treated the same as the *pathname* `/`, the functionality required by Draft 10 meets
 1541 all of the Draft 11 requirements.

4.3 `bc` — Arbitrary-precision arithmetic language

4.3.1 Synopsis

`bc [-l] [file ...]`

4.3.2 Description

The `bc` utility shall implement an arbitrary precision calculator. It shall take input from any files given, then read from the standard input. If the standard input and standard output to `bc` are attached to a terminal, the invocation of `bc` shall be considered to be *interactive*, causing behavioral constraints described in the following subclauses.

4.3.3 Options

The `bc` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following option shall be supported by the implementation:

<p>−l</p>	(The letter ell.) Define the math functions and initialize <code>scale</code> to 20, instead of the default zero. See 4.3.7.
-----------	--

4.3.4 Operands

The following operands shall be supported by the implementation:

<p><i>file</i></p>	A pathname of a text file containing <code>bc</code> program statements. After all <i>files</i> have been read, <code>bc</code> shall read the standard input.
--------------------	--

4.3.5 External Influences

4.3.5.1 Standard Input

See Input Files.

4.3.5.2 Input Files

Input files shall be text files containing a sequence of comments, statements, and function definitions that shall be executed as they are read.

4.3.5.3 Environment Variables

The following environment variables shall affect the execution of `bc`:

LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).
LC_MESSAGES	This variable shall determine the language in which messages should be written.

4.3.5.4 Asynchronous Events

Default.

4.3.6 External Effects

4.3.6.1 Standard Output

The output of the `bc` utility shall be controlled by the program read, and shall consist of zero or more lines containing the value of all executed expressions without assignments. The radix and precision of the output shall be controlled by the values of the `obase` and `scale` variables. See 4.3.7.

4.3.6.2 Standard Error

Used only for diagnostic messages.

4.3.6.3 Output Files

None.

4.3.7 Extended Description

4.3.7.1 bc Grammar

The grammar in this subclause and the lexical conventions in the following subclause shall together describe the syntax for bc programs. The general conventions for this style of grammar are described in 2.1.2. A valid program can be represented as the nonterminal symbol `program` in the grammar. Any discrepancies found between this grammar and other descriptions in this subclause (4.3.7) shall be resolved in favor of this grammar.

```

1603 %token   EOF NEWLINE STRING LETTER NUMBER
1604 %token   MUL_OP
1605 /*      '*' , '/' , '%'                               */
1606 %token   ASSIGN_OP
1607 /*      '=' , '+=' , '-=' , '*=' , '/=' , '%=' , '^=' */
1608 %token   REL_OP
1609 /*      '==', '<=', '>=', '!=', '<', '>'                */
1610 %token   INCR_DECR
1611 /*      '++', '--'                                     */
1612 %token   Define   Break   Quit   Length
1613 /*      'define', 'break', 'quit', 'length'           */
1614 %token   Return   For     If     While   Sqrt
1615 /*      'return', 'for', 'if', 'while', 'sqrt'       */
1616 %token   Scale    Ibase   Obase   Auto
1617 /*      'scale', 'ibase', 'obase', 'auto'            */
1618 %start   program
1619 %%
1620 program      : EOF
1621               | input_item program
1622               ;
1623 input_item    : semicolon_list NEWLINE
1624               | function
1625               ;
1626 semicolon_list : /* empty */
1627               | statement
1628               | semicolon_list ';' statement
1629               | semicolon_list ';'
1630               ;
1631 statement_list : /* empty */
1632               | statement
1633               | statement_list NEWLINE
1634               | statement_list NEWLINE statement
1635               | statement_list ';'
1636               | statement_list ';' statement
1637               ;

```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.


```

1638 statement          : expression
1639                       | STRING
1640                       | Break
1641                       | Quit
1642                       | Return
1643                       | Return '(' return_expression ')'
1644                       | For '(' expression ';'
1645                           relational_expression ';'
1646                           expression ')' statement
1647                       | If '(' relational_expression ')' statement
1648                       | While '(' relational_expression ')' statement
1649                       | '{' statement_list '}'
1650                       ;
1651 function             : Define LETTER '(' opt_parameter_list ')'
1652                       '{' NEWLINE opt_auto_define_list
1653                       statement_list '}'
1654                       ;
1655 opt_parameter_list   : /* empty */
1656                       | parameter_list
1657                       ;
1658 parameter_list       : LETTER
1659                       | define_list ',' LETTER
1660                       ;
1661 opt_auto_define_list : /* empty */
1662                       | Auto define_list NEWLINE
1663                       | Auto define_list ';'
1664                       ;
1665 define_list          : LETTER
1666                       | LETTER '[' ']'
1667                       | define_list ',' LETTER
1668                       | define_list ',' LETTER '[' ']'
1669                       ;
1670 opt_argument_list    : /* empty */
1671                       | argument_list
1672                       ;
1673 argument_list        : expression
1674                       | argument_list ',' expression
1675                       ;
1676 relational_expression : expression
1677                       | expression REL_OP expression
1678                       ;
1679 return_expression    : /* empty */
1680                       | expression
1681                       ;
1682 expression           : named_expression
1683                       | NUMBER
1684                       | '(' expression ')'
1685                       | LETTER '(' opt_argument_list ')'
1686                       | '-' expression
1687                       | expression '+' expression

```

1

```

1688         | expression '-' expression                                1
1689         | expression MUL_OP expression
1690         | expression '^' expression
1691         | INCR_DECR named_expression
1692         | named_expression INCR_DECR
1693         | named_expression ASSIGN_OP expression
1694         | Length '(' expression ')'
1695         | Sqrt '(' expression ')'
1696         | Scale '(' expression ')'
1697         ;
1698 named_expression : LETTER
1699                 | LETTER '[' expression ']'
1700                 | Scale
1701                 | Ibase
1702                 | Obase
1703                 ;

```

1704 4.3.7.2 bc Lexical Conventions

1705 The lexical conventions for bc programs, with respect to the preceding grammar,
 1706 shall be as follows:

- 1707 (1) Except as noted, bc shall recognize the longest possible token or delim-
 1708 iter beginning at a given point.
- 1709 (2) A comment shall consist of any characters beginning with the two adja-
 1710 cent characters /* and terminated by the next occurrence of the two adja-
 1711 cent characters */. Comments shall have no effect except to delimit lexi-
 1712 cal tokens.
- 1713 (3) The character <newline> shall be recognized as the token NEWLINE.
- 1714 (4) The token STRING shall represent a string constant; it shall consist of any
 1715 characters beginning with the double-quote character (") and terminated
 1716 by another occurrence of the double-quote character. The value of the
 1717 string shall be the sequence of all characters between, but not including,
 1718 the two double-quote characters. All characters shall be taken literally
 1719 from the input, and there is no way to specify a string containing a
 1720 double-quote character. The length of the value of each string shall be
 1721 limited to {BC_STRING_MAX} bytes.
- 1722 (5) A <blank> shall have no effect except as an ordinary character if it 1
 1723 appears within a STRING token, or to delimit a lexical token other than 1
 1724 STRING. 1
- 1725 (6) The combination of a backslash character immediately followed by a 2
 1726 <newline> character shall delimit lexical tokens with the following 2
 1727 exceptions: 2
 - 1728 — It shall be interpreted as a literal <newline> in STRING tokens. 2
 - 1729 — It shall be ignored as part of a multiline NUMBER token. 2
- 1730 (7) The token NUMBER shall represent a numeric constant. It shall be recog-
 1731 nized by the following grammar:

Copyright © 1991 IEEE. All rights reserved.
 This is an unapproved IEEE Standards Draft, subject to change.

```

1732         NUMBER    : integer
1733                     | '.' integer
1734                     | integer '.'
1735                     | integer '.' integer
1736                     ;
1737
1737         integer    : digit
1738                     | integer digit
1739                     ;
1740
1740         digit      : 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
1741                     | 8 | 9 | A | B | C | D | E | F
1742                     ;

```

- (8) The value of a `NUMBER` token shall be interpreted as a numeral in the base specified by the value of the internal register `ibase` (described below). Each of the `digit` characters shall have the value from 0 to 15 in the order listed here, and the period character shall represent the radix point. The behavior is undefined if digits greater than or equal to the value of `ibase` appear in the token. (However, note the exception for single-digit values being assigned to `ibase` and `obase` themselves, in 4.3.7.3).

- (9) The following keywords shall be recognized as tokens:

```

1752         auto      for      length  return  sqrt
1753         break     ibase    obase    scale   while
1754         define    if       quit

```

- (10) Any of the following characters occurring anywhere except within a keyword shall be recognized as the token `LETTER`:

```

1757         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

```

- (11) The following single-character and two-character sequences shall be recognized as the token `ASSIGN_OP`:

```

1760         =      +=      -=      *=      /=      %=      ^=

```

- (12) If an `=` character, as the beginning of a token, is followed by a `-` character with no intervening delimiter, the behavior is undefined.

- (13) The following single-characters shall be recognized as the token `MUL_OP`:

```

1764         *      /      %

```

- (14) The following single-character and two-character sequences shall be recognized as the token `REL_OP`:

```

1767         ==      <=      >=      !=      <      >

```

- (15) The following two-character sequences shall be recognized as the token `INCR_DECR`:

```

1770         ++      --

```

(16) The following single characters shall be recognized as tokens whose names are the character:

<newline> () , + - ; [] ^ { } 1

(17) The token EOF shall be returned when the end of input is reached.

4.3.7.3 bc Operations

There are three kinds of identifiers: ordinary identifiers, array identifiers, and function identifiers. All three types consist of single lowercase letters. Array identifiers shall be followed by square brackets ([]). An array subscript is required except in an argument or auto list. Arrays are singly dimensioned and can contain up to {BC_DIM_MAX} elements. Indexing begins at zero so an array is indexed from 0 to {BC_DIM_MAX}–1. Subscripts shall be truncated to integers. Function identifiers shall be followed by parentheses, possibly enclosing arguments. The three types of identifiers do not conflict.

Table 4-3 summarizes the rules for precedence and associativity of all operators. Operators on the same line shall have the same precedence; rows are in order of decreasing precedence.

Table 4-3 – bc Operators

Operator	Associativity
++, --	not applicable
unary -	not applicable
^	right to left
*, /, %	left to right
+, binary -	left to right
=, +=, -=, *=, /=, %=, ^=	right to left
==, <=, >=, !=, <, >	none

Each expression or named expression has a *scale*, which is the number of decimal digits that shall be maintained as the fractional portion of the expression.

Named expressions are places where values are stored. Named expressions shall be valid on the left side of an assignment. The value of a named expression shall be the value stored in the place named. Simple identifiers and array elements shall be named expressions; they shall have an initial value of zero and an initial scale of zero.

The internal registers *scale*, *ibase*, and *obase* are all named expressions. The scale of an expression consisting of the name of one of these registers shall be zero; values assigned to any of these registers shall be truncated to integers. The *scale* register shall contain a global value used in computing the scale of expressions (as described below). The value of the register *scale* shall be limited to $0 \leq \text{scale} \leq \{\text{BC_SCALE_MAX}\}$ and shall have a default value of zero. The *ibase* and *obase* registers are the input and output number radix, respectively. The value of *ibase* shall be limited to

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1813 $2 \leq \text{ibase} \leq 16$

1814 The value of `obase` shall be limited to

1815 $2 \leq \text{obase} \leq \{\text{BC_BASE_MAX}\}$

1816 When either `ibase` or `obase` is assigned a single digit value from the list in
 1817 4.3.7.2, the value shall be assumed in hexadecimal. (For example, `ibase=A` sets
 1818 to base ten, regardless of the current `ibase` value.) Otherwise, the behavior is
 1819 undefined when digits greater than or equal to the value of `ibase` appear in the
 1820 input. Both `ibase` and `obase` shall have initial values of 10.

1821 Internal computations shall be conducted as if in decimal, regardless of the input 1
 1822 and output bases, to the specified number of decimal digits. When an exact result
 1823 is not achieved, (e.g., `scale=0; 3.2/1`) the result shall be truncated.

1824 For all values of `obase` specified by this standard, numerical values shall be out-
 1825 put as follows:

- 1826 (1) If the value is less than zero, a hyphen (-) character shall be output.
- 1827 (2) One of the following shall be output, depending on the numerical value:
 - 1828 — If the absolute value of the numerical value is greater than or equal to
 1829 one, the integer portion of the value shall be output as a series of
 1830 digits appropriate to `obase` (as described below). The most significant
 1831 nonzero digit shall be output next, followed by each successively less
 1832 significant digit.
 - 1833 — If the absolute value of the numerical value is less than one but
 1834 greater than zero and the scale of the numerical value is greater than
 1835 zero, it is unspecified whether the character 0 is output.
 - 1836 — If the numerical value is zero, the character 0 shall be output.
- 1837 (3) If the scale of the value is greater than zero, a period character shall be
 1838 output, followed by a series of digits appropriate to `obase` (as described
 1839 below) representing the most significant portion of the fractional part of
 1840 the value. If `s` represents the scale of the value being output, the number
 1841 of digits output shall be `s` if `obase` is 10, less than or equal to `s` if `obase`
 1842 is greater than 10, or greater than or equal to `s` if `obase` is less than 10.
 1843 For `obase` values other than 10, this should be the number of digits
 1844 needed to represent a precision of 10^s .

1845 For `obase` values from 2 to 16, valid digits are the first `obase` of the single char-
 1846 acters

1847	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

1848 which represent the values zero through fifteen, respectively.

1849 For bases greater than 16, each “digit” shall be written as a separate multidigit
 1850 decimal number. Each digit except the most significant fractional digit shall be
 1851 preceded a single `<space>` character. For bases from 17 to 100, `bc` shall write
 1852 two-digit decimal numbers; for bases from 101 to 999, three-digit decimal strings,
 1853 and so on. For example, the decimal number 1024 in base 25 would be written as:

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1854 $\Delta 01\Delta 15\Delta 24$

1855 in base 125, as:

1856 $\Delta 008\Delta 024$

1857 Very large numbers shall be split across lines with 70 characters per line in the
 1858 POSIX Locale; other locales may split at different character boundaries. Lines
 1859 that are continued shall end with a backslash (\).

1860 A function call shall consist of a function name followed by parentheses contain-
 1861 ing a comma-separated list of expressions, which are the function arguments. A
 1862 whole array passed as an argument shall be specified by the array name followed
 1863 by empty square brackets. All function arguments shall be passed by value. As a
 1864 result, changes made to the formal parameters have no effect on the actual argu-
 1865 ments. If the function terminates by executing a `return` statement, the value of
 1866 the function shall be the value of the expression in the parentheses of the `return`
 1867 statement or shall be zero if no expression is provided or if there is no `return`
 1868 statement.

1869 The result of `sqrt(expression)` shall be the square root of the expression. The
 1870 result shall be truncated in the least significant decimal place. The scale of the
 1871 result shall be the scale of the expression or the value of `scale`, whichever is
 1872 larger.

1873 The result of `length(expression)` shall be the total number of significant decimal
 1874 digits in the expression. The scale of the result shall be zero.

1875 The result of `scale(expression)` shall be the scale of the expression. The scale of
 1876 the result shall be zero.

1877 A numeric constant shall be an expression. The scale shall be the number of
 1878 digits that follow the radix point in the input representing the constant, or zero if
 1879 no radix point appears.

1880 The sequence (*expression*) shall be an expression with the same value and scale
 1881 as *expression*. The parentheses can be used to alter the normal precedence.

1882 The semantics of the unary and binary operators are as follows.

1883 *-expression*

1884 The result shall be the negative of the *expression*. The scale of the
 1885 result shall be the scale of *expression*.

1886 The unary increment and decrement operators shall not modify the scale of the
 1887 named expression upon which they operate. The scale of the result shall be the
 1888 scale of that named expression.

1889 *++named-expression*

1890 The named expression shall be incremented by one. The result shall
 1891 be the value of the named expression after incrementing.

1892 *--named-expression*

1893 The named expression shall be decremented by one. The result shall
 1894 be the value of the named expression after decrementing.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1895 *named-expression*++
 1896 The named expression shall be incremented by one. The result shall
 1897 be the value of the named expression before incrementing.

1898 *named-expression*--
 1899 The named expression shall be decremented by one. The result shall
 1900 be the value of the named expression before decrementing.

1901 The exponentiation operator, circumflex (^), shall bind right to left.

1902 *expression* ^ *expression*
 1903 The result shall be the first *expression* raised to the power of the
 1904 second *expression*. If the second expression is not an integer, the
 1905 behavior is undefined. If *a* is the scale of the left expression and *b* is
 1906 the absolute value of the right expression, the scale of the result
 1907 shall be:

1908 if *b* >= 0 min(*a* * *b*, max(scale, *a*)) 2
 1909 if *b* < 0 scale 2

1910 The multiplicative operators (*, /, %) shall bind left to right.

1911 *expression* * *expression*
 1912 The result shall be the product of the two expressions. If *a* and *b* are
 1913 the scales of the two expressions, then the scale of the result shall be:

1914 min(*a*+*b*,max(scale,*a*,*b*))

1915 *expression* / *expression*
 1916 The result shall be the quotient of the two expressions. The scale of
 1917 the result shall be the value of scale.

1918 *expression* % *expression*
 1919 For expressions *a* and *b*, *a* % *b* shall be evaluated equivalent to the
 1920 steps:

1921 (1) Compute *a*/*b* to current scale.

1922 (2) Use the result to compute

1923 *a* - (*a* / *b*) * *b*

1924 to scale

1925 max(scale + scale(*b*), scale(*a*))

1926 The scale of the result shall be

1927 max(scale + scale(*b*), scale(*a*))

1928 The additive operators (+, -) shall bind left to right.

1929 *expression* + *expression*
 1930 The result shall be the sum of the two expressions. The scale of the
 1931 result shall be the maximum of the scales of the expressions.

1932 *expression* - *expression*
 1933 The result shall be the difference of the two expressions. The scale of

1934 the result shall be the maximum of the scales of the expressions.

1935 The assignment operators ($=$, $+=$, $-=$, $*=$, $/=$, $\%=$, $\hat{=}$) shall bind right to left.

1936 *named-expression* = *expression*

1937 This expression results in assigning the value of the expression on

1938 the right to the named expression on the left. The scale of both the

1939 named expression and the result shall be the scale of *expression*.

1940 The compound assignments forms

1941 *named-expression* <operator>= *expression*

1942 shall be equivalent to:

1943 *named-expression* = *named-expression* <operator> *expression*

1944 except that the *named-expression* shall be evaluated only once.

1945 Unlike all other operators, the relational operators ($<$, $>$, $<=$, $>=$, $==$, $!=$) shall be

1946 only valid as the object of an `if`, `while`, or inside a `for` statement.

1947 *expression1* < *expression2*

1948 The relation shall be true if the value of *expression1* is strictly less

1949 than the value of *expression2*.

1950 *expression1* > *expression2*

1951 The relation shall be true if the value of *expression1* is strictly

1952 greater than the value of *expression2*.

1953 *expression1* <= *expression2*

1954 The relation shall be true if the value of *expression1* is less than or

1955 equal to the value of *expression2*.

1956 *expression1* >= *expression2*

1957 The relation shall be true if the value of *expression1* is greater than

1958 or equal to the value of *expression2*.

1959 *expression1* == *expression2*

1960 The relation shall be true if the values of *expression1* and *expression2*

1961 are equal.

1962 *expression1* != *expression2*

1963 The relation shall be true if the values of *expression1* and *expression2*

1964 are unequal.

1965 There are only two storage classes in `bc`, `global` and `automatic (local)`. Only

1966 identifiers that are to be local to a function need be declared with the `auto` com-

1967 mand. The arguments to a function shall be local to the function. All other

1968 identifiers are assumed to be `global` and available to all functions. All identifiers,

1969 `global` and `local`, have initial values of zero. Identifiers declared as `auto` shall be

1970 allocated on entry to the function and released on returning from the function.

1971 They therefore do not retain values between function calls. `Auto` arrays shall be

1972 specified by the array name followed by empty square brackets. On entry to a

1973 function, the old values of the names that appear as parameters and as `automatic`

1974 variables are pushed onto a stack. Until return is made from the function,

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1975 reference to these names refers only to the new values.

1976 References to any of these names from other functions that are called from this
 1977 function also refer to the new value until one of those functions uses the same
 1978 name for a local variable.

1979 When a statement is an expression, unless the main operator is an assignment,
 1980 execution of the statement shall write the value of the expression followed by a
 1981 <newline> character.

1982 When a statement is a string, execution of the statement shall write the value of
 1983 the string.

1984 Statements separated by semicolon or <newline> shall be executed sequentially.
 1985 In an interactive invocation of `bc`, each time a <newline> character is read that
 1986 satisfies the grammatical production

1987 `input_item : semicolon_list NEWLINE`

1988 the sequential list of statements making up the `semicolon_list` shall be exe-
 1989 cuted immediately and any output produced by that execution shall be written
 1990 without any delay due to buffering.

1991 In an `if` statement [`if (relation) statement`] the *statement* shall be executed if the
 1992 relation is true.

1993 The `while` statement [`while (relation) statement`] implements a loop in which the
 1994 *relation* is tested; each time the *relation* is true, the *statement* shall be executed
 1995 and the *relation* retested. When the *relation* is false, execution shall resume after
 1996 *statement*.

1997 A `for` statement [`for (expression; relation; expression) statement`] shall be the
 1998 same as:

1999 *first-expression*
 2000 `while (relation) {`
 2001 *statement*
 2002 *last-expression*
 2003 `}`

2004 All three expressions shall be present.

2005 The `break` statement causes termination of a `for` or `while` statement.

2006 The `auto` statement [`auto identifier[, identifier] ...`] shall cause the values of the
 2007 identifiers to be pushed down. The identifiers can be ordinary identifiers or array
 2008 identifiers. Array identifiers shall be specified by following the array name by
 2009 empty square brackets. The `auto` statement shall be the first statement in a
 2010 function definition.

2011 A define statement:

```
2012     define LETTER ( opt_parameter_list ) {
2013         opt_auto_define_list
2014         statement_list
2015     }
```

2016 defines a function named *LETTER*. If a function named *LETTER* was previously
2017 defined, the define statement shall replace the previous definition. The expres-
2018 sion

```
2019     LETTER ( opt_argument_list )
```

2020 shall invoke the function named *LETTER*. The behavior is undefined if the
2021 number of arguments in the invocation does not match the number of parameters
2022 in the definition. Functions shall be defined before they are invoked. A function
2023 shall be considered to be defined within its own body, so recursive calls shall be
2024 valid. The values of numeric constants within a function shall be interpreted in
2025 the base specified by the value of the *ibase* register when the function is invoked.

2026 The return statements [*return* and *return(expression)*] shall cause termina-
2027 tion of a function, popping of its auto variables, and specifies the result of the
2028 function. The first form shall be equivalent to *return(0)*. The value and scale
2029 of an invocation of the function shall be the value and scale of the expression in
2030 parentheses.

2031 The quit statement (*quit*) shall stop execution of a *bc* program at the point
2032 where the statement occurs in the input, even if it occurs in a function definition,
2033 or in an *if*, *for*, or *while* statement.

2034 The following functions shall be defined when the *-l* option is specified:

```
2035     s ( Expression )      Sine of argument in radians
2036     c ( Expression )      Cosine of argument in radians
2037     a ( Expression )      Arctangent of argument
2038     l ( Expression )      Natural logarithm of argument
2039     e ( Expression )      Exponential function of argument
2040     j ( Expression , Expression )
2041                               Bessel function of integer order
```

2042 The scale of an invocation of each of these functions shall be the value of the
2043 scale register when the function is invoked. The behavior is undefined if any of
2044 these functions is invoked with an argument outside the domain of the mathemat-
2045 ical function.

4.3.8 Exit Status

The `bc` utility shall exit with one of the following values:

0	All input files were processed successfully.
<i>unspecified</i>	An error occurred.

4.3.9 Consequences of Errors

If any *file* operand is specified and the named file cannot be accessed, `bc` shall write a diagnostic message to standard error and terminate without any further action.

In an interactive invocation of `bc`, the utility should print an error message and recover following any error in the input. In a noninteractive invocation of `bc`, invalid input causes undefined behavior.

4.3.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

This description is based on *BC—An Arbitrary Precision Desk-Calculator Language* by Lorinda Cherry and Robert Morris, in the BSD User Manual {B28}.

Automatic variables in `bc` do not work in exactly the same way as in either C or PL/1.

In the shell, the following assigns an approximation of the first ten digits of π to the variable `x`:

```
x=$(printf "%s\n" 'scale = 10; 104348/33215' | bc)
```

The following `bc` program prints the same approximation of π , with a label, to standard output:

```
scale = 10
"pi equals "
104348 / 33215
```

The following defines a function to compute an approximate value of the exponential function (note that such a function is predefined if the `-l` option is specified):

```

2073     scale = 20
2074     define e(x){
2075         auto a, b, c, i, s
2076         a = 1
2077         b = 1
2078         s = 1
2079         for (i = 1; 1 == 1; i++){
2080             a = a*x
2081             b = b*i
2082             c = a/b
2083             if (c == 0) {
2084                 return(s)
2085             }
2086             s = s+c
2087         }
2088     }

```

2089 The following prints approximate values of the exponential function of the first
 2090 ten integers:

```

2091     for (i = 1; i <= 10; ++i) {
2092         e(i)
2093     }

```

2094 History of Decisions Made

2095 The `bc` utility is traditionally implemented as a front-end processor for `dc`; `dc`
 2096 was not selected to be part of the standard because `bc` was thought to have a
 2097 more intuitive programmatic interface. Current implementations that implement
 2098 `bc` using `dc` are expected to be compliant.

2099 The Exit Status for error conditions been left unspecified for several reasons:

- 2100 (1) The `bc` utility is used in both interactive and noninteractive situations.
 2101 Different exit codes may be appropriate for the two uses.
- 2102 (2) It is unclear when a nonzero exit should be given; divide-by-zero,
 2103 undefined functions, and syntax errors are all possibilities.
- 2104 (3) It is not clear what utility the exit status has.
- 2105 (4) In the 4.3BSD, System V, and Ninth Edition implementations, `bc` works
 2106 in conjunction with `dc`. `dc` is the parent, `bc` is the child. This was done
 2107 to cleanly terminate `bc` if `dc` aborted.

2108 The decision to have `bc` exit upon encountering an inaccessible input file is based
 2109 on the belief that `bc file1 file2` is used most often when at least *file1* contains
 2110 data/function declarations/initializations. Having `bc` continue with prerequisite
 2111 files missing is probably not useful. There is no implication in the Consequences
 2112 of Errors subclause that `bc` must check all its files for accessibility before opening
 2113 any of them.

2114 There was considerable debate on the appropriateness of the language accepted
 2115 by `bc`. Several members of the balloting group preferred to see either a pure

subset of the C language or some changes to make the language more compatible with C. While the `bc` language has some obvious similarities to C, it has never claimed to be compatible with any version of C. An interpreter for a subset of C might be a very worthwhile utility, and it could potentially make `bc` obsolete. However, no such utility is known in existing practice, and it was not within the scope of POSIX.2 to define such a language and utility. If and when they are defined, it may be appropriate to include them in a future revision of this standard. This left the following alternatives:

- (1) Exclude any calculator language from the standard.

The consensus of the working group was that a simple programmatic calculator language is very useful. Also, an interactive version of such a calculator would be very important for the POSIX.2a revision. The only arguments for excluding any calculator were that it would become obsolete if and when a C-compatible one emerged, or that the absence would encourage the development of such a C-compatible one. These arguments did not sufficiently address the needs of current application writers.

- (2) Standardize the existing `dc`, possibly with minor modifications.

The consensus of the working group was that `dc` is a fundamentally less usable language and that that would be far too severe a penalty for avoiding the issue of being similar to but incompatible with C.

- (3) Standardize the existing `bc`, possibly with minor modifications.

This was the approach taken. Most of the proponents of changing the language would not have been satisfied until most or all of the incompatibilities with C were resolved. Since most of the changes considered most desirable would break existing applications and require significant modification to existing implementations, almost no modifications were made. The one significant modification that was made was the replacement of the traditional `bc`'s assignment operators `=+ et al.` with the more modern `+= et al.` The older versions are considered to be fundamentally flawed because of the lexical ambiguity in uses like

```
a=-1
```

In order to permit implementations to deal with backward compatibility as they see fit, the behavior of this one ambiguous construct was made undefined. (At least three implementations have been known to support this change already, so the degree of change involved should not be great.)

The `%` operator is the mathematical remainder operator when `scale` is zero. The behavior of this operator for other values of `scale` is from traditional implementations of `bc`, and has been maintained for the sake of existing applications despite its nonintuitive nature.

The `bc` utility always uses the period (`.`) character to represent a radix point, regardless of any decimal-point character specified as part of the current locale.

In languages like C or awk, the period character is used in program source, so it can be portable and unambiguous, while the locale-specific character is used in input and output. Because there is no distinction between source and input in bc, this arrangement would not be possible. Using the locale-specific character in bc's input would introduce ambiguities into the language; consider the following example in a locale with a comma as the decimal-point character:

```
define f(a,b) {
    ...
}
...
f(1,2,3)
```

Because of such ambiguities, the period character is used in input. Having input follow different conventions from output would be confusing in either pipeline usage or interactive usage, so period is also used in output.

Traditional implementations permit setting `ibase` and `obase` to a broader range of values. This includes values less than 2, which were not seen as sufficiently useful to standardize. These implementations do not interpret input properly for values of `ibase` outside greater than 16. This is because numeric constants are recognized syntactically, rather than lexically, as described in the standard. They are built from lexical tokens of single hexadecimal digits and periods. Since `<blank>s` between tokens are not visible at the syntactic level, it is not possible to properly recognize the multidigit “digits” used in the higher bases. The ability to recognize input in these bases was not considered useful enough to require modifying these implementations. Note that the recognition of numeric constants at the syntactic level is not a problem with conformance to the standard, as it does not impact the behavior of portable applications (and correct bc programs). Traditional implementations also accept input with all of the digits 0-9 and A-F regardless of the value of `ibase`; since digits with value greater than or equal to `ibase` are not really appropriate, the behavior when they appear is undefined, except for the common case of

```
ibase=8;
/* Process in octal base */
...
ibase=A
/* Restore decimal base */
```

In some historical implementations, if the expression to be written is an uninitialized array element, a leading `<space>` character and/or up to four leading 0 characters may be output before the character zero. This behavior is considered a bug; it is unlikely that any currently portable application relies on

```
echo 'b[3]' | bc
```

returning 00000 rather than 0.

Exact calculation of the number of fractional digits to output for a given value in a base other than 10 can be computationally expensive. Traditional implementations use a faster approximation, and this is permitted. Note that the requirements apply only to values of `obase` that the standard requires implementations

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2204 to support (in particular, not to 1, 0, or negative bases, if an implementation sup-
 2205 ports them as an extension).

2206 **4.4 cat — Concatenate and print files**

2207 **4.4.1 Synopsis**

2208 `cat [-u] [file ...]`

2209 **4.4.2 Description**

2210 The `cat` utility reads files in sequence and writes their contents to the standard
 2211 output in the same sequence.

2212 **4.4.3 Options**

2213 The `cat` utility shall conform to the utility argument syntax guidelines described
 2214 in 2.10.2.

2215 The following option shall be supported by the implementation:

2216 `-u` Write bytes from the input file to the standard output without
 2217 delay as each is read.

2218 **4.4.4 Operands**

2219 The following operand shall be supported by the implementation:

2220 *file* A pathname of an input file. If no *file* operands are specified, the
 2221 standard input is used. If a *file* is `-`, the `cat` utility shall read
 2222 from the standard input at that point in the sequence. The `cat`
 2223 utility shall not close and reopen standard input when it is refer-
 2224 enced in this way, but shall accept multiple occurrences of `-` as a
 2225 *file* operand.

2226 **4.4.5 External Influences**

2227 **4.4.5.1 Standard Input**

2228 The standard input is used only if no *file* operands are specified, or if a *file*
 2229 operand is `-`. See Input Files.

4.4.5.2 Input Files

The input files can be any file type.

4.4.5.3 Environment Variables

The following environment variables shall affect the execution of `cat`:

LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).
LC_MESSAGES	This variable shall determine the language in which messages should be written.

4.4.5.4 Asynchronous Events

Default.

4.4.6 External Effects**4.4.6.1 Standard Output**

The standard output shall contain the sequence of bytes read from the input file(s). Nothing else shall be written to the standard output.

4.4.6.2 Standard Error

Used only for diagnostic messages.

4.4.6.3 Output Files

None.

2257 **4.4.7 Extended Description**

2258 None.

2259 **4.4.8 Exit Status**

2260 The `cat` utility shall exit with one of the following values:

2261 0 All input files were output successfully.

2262 >0 An error occurred.

2263 **4.4.9 Consequences of Errors**

2264 Default.

2265 **4.4.10 Rationale.** *(This subclause is not a part of P1003.2)*

2266 **Examples, Usage**

2267 Historical versions of the `cat` utility include the options `-e`, `-t`, and `-v`, which
2268 permit the ends of lines, `<tab>s`, and invisible characters, respectively, to be ren-
2269 dered visible in the output. The working group omitted these options because
2270 they provide too fine a degree of control over what is made visible, and similar
2271 output can be obtained using a command such as:

2272 `sed -n -e 's/$$/$/' -e l pathname`

2273 The `-s` option was omitted because it corresponds to different functions in BSD
2274 and System V-based systems. The BSD `-s` option to squeeze blank lines will be
2275 handled by `more -s` in the UPE. The System V `-s` option to silence error mes-
2276 sages can be accomplished by redirecting the standard error. An alternative to
2277 `cat -s` is the following shell script using `sed`:

```

2278     sed -n '
2279     # Write non-empty lines.
2280     ./ {
2281         p
2282         d
2283     }
2284     # Write a single empty line, then look for more empty lines.
2285     /^$/ p
2286     # Get next line, discard the held <newline> (empty line),
2287     # and look for more empty lines.
2288     :Empty
2289     /^$/ {
2290         N
2291         s/./ /
2292         b Empty
2293     }
2294     # Write the non-empty line before going back to search
2295     # for the first in a set of empty lines.
2296     p
2297     '

```

2298 Note that the BSD documentation for `cat` uses the term “blank line” to mean the
 2299 same as the POSIX “empty line”; a line consisting only of a `<newline>`.

2300 The BSD `-n` option is omitted because similar functionality can be obtained from
 2301 the `-n` option of the `pr` utility.

2302 The `-u` option is included here for its value in prototyping nonblocking reads from
 2303 FIFOs. The intent is to support the following sequence:

```

2304     mkfifo foo
2305     cat -u foo > /dev/tty13 &
2306     cat -u > foo

```

2307 It is unspecified whether standard output is or is not buffered in the default case.
 2308 This is sometimes of interest when standard output is associated with a terminal,
 2309 since buffering may delay the output. The presence of the `-u` option guarantees
 2310 that unbuffered I/O is available. It is implementation dependent whether the `cat`
 2311 utility buffers output if the `-u` option is not specified. Traditionally, the `-u` option
 2312 is implemented using the BSD `setbuffer()` function, the System V `setbuf()` function,
 2313 or the C Standard {7} `setvbuf()` function.

2314 The following command

```
2315     cat myfile
```

2316 writes the contents of the file `myfile` to standard output.

2317 The following command

```
2318     cat doc1 doc2 > doc.all
```

2319 concatenates the files `doc1` and `doc2` and writes the result to `doc.all`.

2320 Because of the shell language mechanism used to perform output redirection, a
 2321 command such as this:

2322 `cat doc doc.end > doc`

2323 causes the original data in `doc` to be lost.

2324 Due to changes made to subclause 2.11.4 in Draft 11, the description of the *file*
 2325 operand now states that `-` must be accepted multiple times, as in historical prac-
 2326 tice. This allows the command:

2327 `cat start - middle - end > file`

2328 when standard input is a terminal, to get two arbitrary pieces of input from the
 2329 terminal with a single invocation of `cat`. Note, however, that if standard input is
 2330 a regular file, this would be equivalent to the command:

2331 `cat start - middle /dev/null end > file`

2332 because the entire contents of the file would be consumed by `cat` the first time `-`
 2333 was used as a *file* operand and an end-of-file condition would be detected immedi-
 2334 ately when `-` was referenced the second time.

2335 **History of Decisions Made**

2336 None.

2337 **4.5 cd — Change working directory**

2338 **4.5.1 Synopsis**

2339 `cd [directory]`

2340 **4.5.2 Description**

2341 The `cd` utility shall change the working directory of the current shell execution
 2342 environment; see 3.12.

2343 When invoked with no operands, and the **HOME** environment variable is set to a
 2344 nonempty value, the directory named in the **HOME** environment variable shall
 2345 become the new working directory. If **HOME** is empty or is undefined, the default
 2346 behavior is implementation defined.

2347 **4.5.3 Options**

2348 None.

2349 4.5.4 Operands

2350 The following operands shall be supported by the implementation:

2351 *directory* An absolute or relative pathname of the directory that becomes
 2352 the new working directory. The interpretation of a relative path-
 2353 name by `cd` depends on the **CDPATH** environment variable. If
 2354 *directory* is `-`, the results are implementation defined.

2355 4.5.5 External Influences

2356 4.5.5.1 Standard Input

2357 None.

2358 4.5.5.2 Input Files

2359 None.

2360 4.5.5.3 Environment Variables

2361 The following environment variables shall affect the execution of `cd`:

2362 **CDPATH** A colon-separated list of pathnames that refer to direc-
 2363 tories. If the *directory* operand does not begin with a
 2364 slash (/) character, and the first component is not dot or
 2365 dot-dot, `cd` shall search for *directory* relative to each direc-
 2366 tory named in the **CDPATH** variable, in the order listed.
 2367 The new working directory shall be set to the first match-
 2368 ing directory found. An empty string in place of a direc-
 2369 tory pathname represents the current directory. If
 2370 **CDPATH** is not set, it shall be treated as if it were an
 2371 empty string.

2372 **HOME** The name of the home directory, used when no *directory*
 2373 operand is specified.

2374 **LANG** This variable shall determine the locale to use for the
 2375 locale categories when both **LC_ALL** and the correspond-
 2376 ing environment variable (beginning with **LC_**) do not
 2377 specify a locale. See 2.6.

2378 **LC_ALL** This variable shall determine the locale to be used to over-
 2379 ride any values for locale categories specified by the set-
 2380 tings of **LANG** or any environment variables beginning
 2381 with **LC_**.

2382 **LC_CTYPE** This variable shall determine the locale for the interpreta-
 2383 tion of sequences of bytes of text data as characters (e.g.,
 2384 single- versus multibyte characters in arguments).

2385 **LC_MESSAGES** This variable shall determine the language in which mes-
 2386 sages should be written.

2387 **4.5.5.4 Asynchronous Events**

2388 Default.

2389 **4.5.6 External Effects**

2390 **4.5.6.1 Standard Output**

2391 If a nonempty directory name from **CDPATH** is used, an absolute pathname of the
 2392 new working directory shall be written to the standard output as follows:

2393 "%s\n", <*new directory*>

2394 Otherwise, there shall be no output.

2395 **4.5.6.2 Standard Error**

2396 Used only for diagnostic messages.

2397 **4.5.6.3 Output Files**

2398 None.

2399 **4.5.7 Extended Description**

2400 None.

2401 **4.5.8 Exit Status**

2402 The **cd** utility shall exit with one of the following values:

2403 0 The directory was successfully changed.

2404 >0 An error occurred.

4.5.9 Consequences of Errors

The working directory remains unchanged.

4.5.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

Editor's Note: A balloter requested that the following rationale be highlighted in the D11.2 recirculation.

Since `cd` affects the current shell execution environment, it is generally provided as a shell regular built-in. If it is called in a subshell or separate utility execution environment, such as one of the following:

```
(cd /tmp)
nohup cd
find . -exec cd {} \;
```

it will not affect the working directory of the caller's environment.

The use of the **CDPATH** was introduced in the System V shell. Its use is analogous to the use of the **PATH** variable in the shell. Earlier systems such as the BSD C-shell used a shell parameter `cdpath` for this purpose.

History of Decisions Made

A common extension when **HOME** is undefined is to get the login directory from the user database for the invoking user. This does not occur on System V implementations.

Not included in this description are the features from the KornShell such as setting **OLDPWD**, toggling current and previous directory (`cd -`), and the two-operand form of `cd` (`cd old new`). This standard does not specify the results of `cd -` or of calls with more than one operand. Since these extensions are mostly used in interactive situations, they may be considered for inclusion in POSIX.2a. The result of `cd -` and of using no arguments with **HOME** unset or null have been made implementation defined at the request of the POSIX.6 security working group.

The setting of the **PWD** variable was removed from earlier drafts, as it can be replaced by `$(pwd)`.

4.6 chgrp — Change file group ownership

4.6.1 Synopsis

chgrp [-R] *group file* ...

4.6.2 Description

The `chgrp` utility shall set the group ID of the file named by each *file* operand to the group ID specified by the *group* operand.

For each *file* operand, it shall perform actions equivalent to the POSIX.1 {8} `chown()` function, called with the following arguments:

- (1) The *file* operand shall be used as the *path* argument.
- (2) The user ID of the file shall be used as the *owner* argument.
- (3) The specified *group ID* shall be used as the *group* argument.

4.6.3 Options

The `chgrp` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following option shall be supported by the implementation:

- | | |
|----|--|
| -R | Recursively change file group IDs. For each <i>file</i> operand that names a directory, <code>chgrp</code> shall change the group of the directory and all files in the file hierarchy below it. |
|----|--|

4.6.4 Operands

The following operands shall be supported by the implementation:

- | | |
|--------------|---|
| <i>group</i> | A group name from the group database or a numeric group ID. Either specifies a group ID to be given to each file named by one of the <i>file</i> operands. If a numeric <i>group</i> operand exists in the group database as a group name, the group ID number associated with that group name is used as the group ID. |
| <i>file</i> | A pathname of a file whose group ID is to be modified. |

2461 **4.6.5 External Influences**

2462 **4.6.5.1 Standard Input**

2463 None.

2464 **4.6.5.2 Input Files**

2465 None.

2466 **4.6.5.3 Environment Variables**

2467 The following environment variables shall affect the execution of `chgrp`:

2468 2469 2470 2471	LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
------------------------------	-------------	---

2472 2473 2474 2475	LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
------------------------------	---------------	--

2476 2477 2478	LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).
----------------------	-----------------	--

2479 2480	LC_MESSAGES	This variable shall determine the language in which messages should be written.
--------------	--------------------	---

2481 **4.6.5.4 Asynchronous Events**

2482 Default.

2483 **4.6.6 External Effects**

2484 **4.6.6.1 Standard Output**

2485 None.

2486 **4.6.6.2 Standard Error**

2487 Used only for diagnostic messages.

4.6.6.3 Output Files

None.

4.6.7 Extended Description

None.

4.6.8 Exit Status

The `chgrp` utility shall exit with one of the following values:

- | | |
|----|--|
| 0 | The utility executed successfully and all requested changes were made. |
| >0 | An error occurred. |

4.6.9 Consequences of Errors

If, when invoked with the `-R` option, `chgrp` attempts but fails to change the group ID of a particular file in a specified file hierarchy, it shall continue to process the remaining files in the hierarchy. If `chgrp` cannot read or search a directory within a hierarchy, it shall continue to process the other parts of the hierarchy that are accessible.

4.6.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it can overflow the range of valid exit status value. The working group chose to mask these by specifying only 0 and >0 as exit values.

History of Decisions Made

The functionality of `chgrp` is described substantially through references to functions in POSIX.1 {8}. In this way, there is no duplication of effort required for describing the interactions of permissions, multiple groups, etc.

4.7 `chmod` — Change file modes

4.7.1 Synopsis

`chmod` `[-R]` *mode file* ...

4.7.2 Description

The `chmod` utility shall change any or all of the file mode bits of the file named by each *file* operand in the way specified by the *mode* operand.

It is implementation defined whether and how the `chmod` utility affects any alternate or additional file access control mechanism (see *file access permissions* in 2.2.2.55) being used for the specified file.

Only a process whose effective user ID matches the user ID of the file, or a process with the appropriate privileges, shall be permitted to change the file mode bits of a file.

4.7.3 Options

The `chmod` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following option shall be supported by the implementation:

<p>−R</p>	<p>Recursively change file mode bits. For each <i>file</i> operand that names a directory, <code>chmod</code> shall change the file mode bits of the directory and all files in the file hierarchy below it.</p>
-----------	--

4.7.4 Operands

The following operands shall be supported by the implementation:

<p><i>mode</i></p>	<p>Represents the change to be made to the file mode bits of each file named by one of the <i>file</i> operands, as described in 4.7.7.</p>
--------------------	---

<p><i>file</i></p>	<p>A pathname of a file whose file mode bits are to be modified.</p>
--------------------	--

4.7.5 External Influences

4.7.5.1 Standard Input

None.

2539 **4.7.5.2 Input Files**

2540 None.

2541 **4.7.5.3 Environment Variables**

2542 The following environment variables shall affect the execution of `chmod`:

2543 2544 2545 2546	LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
------------------------------	-------------	---

2547 2548 2549 2550	LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
------------------------------	---------------	--

2551 2552 2553	LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).
----------------------	-----------------	--

2554 2555	LC_MESSAGES	This variable shall determine the language in which messages should be written.
--------------	--------------------	---

2556 **4.7.5.4 Asynchronous Events**

2557 Default.

2558 **4.7.6 External Effects**

2559 **4.7.6.1 Standard Output**

2560 None.

2561 **4.7.6.2 Standard Error**

2562 Used only for diagnostic messages.

2563 **4.7.6.3 Output Files**

2564 None.

4.7.7 Extended Description

The *mode* operand shall be either a `symbolic_mode` expression or a nonnegative octal integer. The `symbolic_mode` form is described by the grammar in 4.7.7.1.

Each `clause` shall specify an operation to be performed on the current file mode bits of each *file*. The operations shall be performed on each *file* in the order in which the `clauses` are specified.

The *who* symbols `u`, `g`, and `o` shall specify the *user*, *group*, and *other* parts of the file mode bits, respectively. A *who* consisting of the symbol `a` shall be equivalent to `ugo`.

The *perm* symbols `r`, `w`, and `x` represent the *read*, *write*, and *execute/search* portions of file mode bits, respectively. The *perm* symbol `s` shall represent the *set-user-ID-on-execution* (when *who* contains or implies `u`) and *set-group-ID-on-execution* (when *who* contains or implies `g`) bits.

The *perm* symbol `X` shall represent the *execute/search* portion of the file mode bits if the file is a directory or if the current (unmodified) file mode bits have at least one of the *execute* bits (`S_IXUSR`, `S_IXGRP`, or `S_IXOTH`) set. It shall be ignored if the file is not a directory and none of the *execute* bits are set in the current file mode bits.

The *permcop* symbols `u`, `g`, and `o` shall represent the current permissions associated with the user, group, and other parts of the file mode bits, respectively. For the remainder of subclause 4.7.7 up to subclause 4.7.7.1, *perm* refers to the non-terminals *perm* and *permcop* in the grammar in 4.7.7.1.

If multiple *actionlists* are grouped with a single *wholist* in the grammar, each *actionlist* shall be applied in the order specified with that *wholist*. The *op* symbols shall represent the operation performed, as follows:

+ If *perm* is not specified, the `+` operation shall not change the file mode bits.

If *who* is not specified, the file mode bits represented by *perm* for the owner, group, and other permissions, except for those with corresponding bits in the file mode creation mask of the invoking process, shall be set.

Otherwise, the file mode bits represented by the specified *who* and *perm* values shall be set.

- If *perm* is not specified, the `-` operation shall not change the file mode bits.

If *who* is not specified, the file mode bits represented by *perm* for the owner, group, and other permissions, except for those with corresponding bits in the file mode creation mask of the invoking process, shall be cleared.

Otherwise, the file mode bits represented by the specified *who* and *perm* values shall be cleared.

- 2606 = Clear the file mode bits specified by the `who` value, or, if no `who` value is
2607 specified, all of the file mode bits specified in this standard.
- 2608 If `perm` is not specified, the `=` operation shall make no further
2609 modifications to the file mode bits.
- 2610 If `who` is not specified, the file mode bits represented by `perm` for the
2611 owner, group, and other permissions, except for those with correspond-
2612 ing bits in the file mode creation mask of the invoking process, shall be
2613 set.
- 2614 Otherwise, the file mode bits represented by the specified `who` and `perm`
2615 values shall be set.
- 2616 When using the symbolic mode form on a regular file, it is implementation defined
2617 whether or not:
- 2618 (1) Requests to set the set-user-ID-on-execution or set-group-ID-on-execution
2619 bit when all execute bits are currently clear and none are being set are
2620 ignored,
 - 2621 (2) Requests to clear all execute bits also clear the set-user-ID-on-execution
2622 and set-group-ID-on-execution bits, or
 - 2623 (3) Requests to clear the set-user-ID-on-execution or set-group-ID-on-
2624 execution bits when all execute bits are currently clear are ignored.
2625 However, if the command `ls -l file` (see 4.39.6.1) writes an `s` in the
2626 positions indicating that the set-user-ID-on-execution or set-group-ID-on-
2627 execution, the commands `chmod u-s file` or `chmod g-s file`, respec-
2628 tively, shall not be ignored.
- 2629 When using the symbolic mode form on other file types, it is implementation 2
2630 defined whether or not requests to set or clear the set-user-ID-on-execution or 2
2631 set-group-ID-on-execution bits are honored. 2
- 2632 If the `who` symbol `o` is used in conjunction with the `perm` symbol `s` with no other
2633 `who` symbols being specified, the set-user-ID-on-execution and set-group-ID-on-
2634 execution bits shall not be modified. It shall not be an error to specify the `who`
2635 symbol `o` in conjunction with the `perm` symbol `s`.
- 2636 For an octal integer *mode* operand, the file mode bits shall be set absolutely. The
2637 octal number form of the *mode* operand is obsolescent.
- 2638 For each bit set in the octal number, the corresponding file permission bit shown 2
2639 in the following table shall be set; all other file permission bits shall be cleared. 2
2640 For regular files, for each bit set in the octal number corresponding to the set- 2
2641 user-ID-on-execution or the set-group-ID-on-execution bits shown in the following 2
2642 table shall be set; if these bits are not set in the octal number, they shall be 2
2643 cleared. For other file types, it is implementation defined whether or not requests 2
2644 to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are 2
2645 honored. 2

2646

2647

2648

2649

2650

Octal	Mode bit	Octal	Mode bit	Octal	Mode bit	Octal	Mode bit
4000	S_ISUID	0400	S_IRUSR	0040	S_IRGRP	0004	S_IROTH
2000	S_ISGID	0200	S_IWUSR	0020	S_IWGRP	0002	S_IWOTH
		0100	S_IXUSR	0010	S_IXGRP	0001	S_IXOTH

2651

When bits are set in the octal number other than those listed in the table above, the behavior is unspecified.

2652

2653

4.7.7.1 chmod Grammar

2654

2655

2656

2657

2658

2659

The grammar and lexical conventions in this subclause describe the syntax for the `symbolic_mode` operand. The general conventions for this style of grammar are described in 2.1.2. A valid `symbolic_mode` can be represented as the nonterminal symbol `symbolic_mode` in the grammar. Any discrepancies found between this grammar and descriptions in the rest of this clause shall be resolved in favor of this grammar.

2660

2661

The lexical processing shall be based entirely on single characters. Implementations need not allow <blank>s within the single argument being processed.

2662

2663

2664

2665

2666

2667

2668

2669

2670

2671

2672

2673

2674

2675

2676

2677

2678

2679

2680

2681

2682

2683

2684

```

%start      symbolic_mode
%%
symbolic_mode : clause
               | symbolic_mode ',' clause
               ;
clause       : actionlist
               | wholist actionlist
               ;
wholist      : who
               | wholist who
               ;
who          : 'u'
               | 'g'
               | 'o'
               | 'a'
               ;
actionlist   : action
               | actionlist action
               ;
action       : op
               | op permlist
               | op permcop
               ;

```

```

2685         permcopu      : 'u'
2686                   | 'g'
2687                   | 'o'
2688                   ;
2689         op              : '+'
2690                   | '-'
2691                   | '='
2692                   ;
2693         permlist        : perm
2694                   | perm permlist
2695                   ;
2696         perm            : 'r'
2697                   | 'w'
2698                   | 'x'
2699                   | 'X'
2700                   | 's'
2701                   ;

```

2702 4.7.8 Exit Status

2703 The `chmod` utility shall exit with one of the following values:

- 2704 0 The utility executed successfully and all requested changes were made.
- 2705 >0 An error occurred.

2706 4.7.9 Consequences of Errors

2707 If, when invoked with the `-R` option, `chmod` attempts but fails to change the mode
 2708 of a particular file in a specified file hierarchy, it shall continue to process the
 2709 remaining files in the hierarchy, affecting the final exit status. If `chmod` cannot
 2710 read or search a directory within a hierarchy, it shall continue to process the
 2711 other parts of the hierarchy that are accessible.

2712 4.7.10 Rationale. *(This subclause is not a part of P1003.2)*

2713 Examples, Usage

2714 The functionality of `chmod` is described substantially through references to con-
 2715 cepts defined in POSIX.1 {8}. In this way, there is less duplication of effort
 2716 required for describing the interactions of permissions, etc. However, the
 2717 behavior of this utility is not described in terms of the `chmod()` function from
 2718 POSIX.1 {8}, because that specification requires certain side effects upon alternate
 2719 file access control mechanisms that might not be appropriate, depending on the
 2720 implementation.

2721 Some historical implementations of the `chmod` utility change the mode of a direc-
 2722 tory before the files in the directory when performing a recursive (`-R` option)

change; others change the directory mode after the files in the directory. If an application tries to remove read or search permission for a file hierarchy, the removal attempt will fail if the directory is changed first; on the other hand, trying to re-enable permissions to a restricted hierarchy will fail if directories are changed last. Since neither method is clearly better and users do not frequently try to make a hierarchy inaccessible to themselves, the standard does not specify what happens in this case.

Note that although the association shown in the table between bits in the octal number and the indicated file mode bits must be supported, this does not require that a conforming implementation has to actually use those octal values to implement the macros shown.

Historical System V implementations of `chmod` never use the process's *umask* when changing modes. Version 7 and historical BSD systems do use the mask when *who* is not specified, as described in this standard. Applications should note the difference between:

```
chmod a-w file
```

which removes all write permissions, and:

```
chmod -- -w file
```

which removes write permissions that would be allowed if *file* was created with the same *umask*. Note that *mode* operands `-r`, `-w`, `-s`, `-x`, or `-X`, or anything beginning with a hyphen, must be preceded by `--` to keep it from being interpreted as an option.

It is difficult to express the grammar used by `chmod` in English, but the following examples have been accepted by historical System V and BSD systems and are, therefore, required to behave this way by POSIX.2 even though some of them could be expressed more succinctly:

<u>Mode</u>	<u>Results</u>
<code>a+=</code>	Equivalent to <code>a+ , a=</code> ; clears all file mode bits.
<code>go+-w</code>	Equivalent to <code>go+ , go-w</code> ; clears group and other write bits.
<code>g=o-w</code>	Equivalent to <code>g=o , g-w</code> ; sets group bit to match other bits and then clears group write bit.
<code>g-r+w</code>	Equivalent to <code>g-r , g+w</code> ; clears group read bit and sets group write bit.
<code>=g</code>	Sets owner bits to match group bits and sets other bits to match group bits.

History of Decisions Made

Implementations that support mandatory file and record locking as specified by the */usr/group Standard* {B29} historically used the combination of set-group-ID bit set and group execute bit clear to indicate mandatory locking. This condition is usually set or cleared with the symbolic mode `perm` symbol `1` instead of the

perm symbols *s* and *x* so that mandatory locking mode is not changed without explicit indication that that was what the user intended. Therefore, the details on how the implementation treats these conditions must be defined in the documentation. This standard does not require mandatory locking (nor does POSIX.1 {8}), but does allow it as an extension. However, POSIX.2 does require that the *ls* and *chmod* utilities work consistently in this area. If *ls -l file* says the set-group-ID bit is set, *chmod g-s file* must clear it (assuming appropriate privileges exist to change modes).

The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it can overflow the range of valid exit status values. This problem is avoided here by specifying only 0 and >0 as exit values.

A “sticky” file mode bit, indicating that the text portion of an executable object program file should be saved after the program is gone, has meaning in some implementations, but was omitted here because its purpose is implementation dependent and because it was omitted from POSIX.1 {8}. On 4.3BSD-based implementations, the sticky bit is used in conjunction with directory permissions to keep anyone from deleting a file that they do not own from the directory. The perm symbol *t* is used to represent the sticky bit in many existing implementations and should not be used for other conflicting extensions.

POSIX.1 {8} indicates that implementation-defined restrictions may cause the *S_ISUID* and *S_ISGID* bits to be ignored. POSIX.2 allows the *chmod* utility to choose to modify these bits before calling POSIX.1 {8} *chmod()* (or some function providing equivalent capabilities) for nonregular files. Among other things, this allows implementations that use the set-user-ID and set-group-ID bits on directories to enable extended features to handle these extensions in an intelligent manner. Portable applications should never assume that they know how these bits will be interpreted, except on regular files.

The grammar in Draft 9 did not allow several symbolic mode operands that are correctly processed by historical implementations. (It only allowed two clauses and one op per clause.) The grammar presented in Draft 10 matches historical implementations.

The *X* perm symbol was added, as provided in BSD-based systems, because it provides commonly desired functionality when doing recursive (*-R* option) modifications. Similar functionality is not provided by the *find* utility. Historical BSD versions of *chmod*, however, only supported *X* with op *+*; it has been extended here because it is also useful with op *=*. (It has also been added for op *-* even though it duplicates *x*, in this case, because it is intuitive and easier to explain.)

The grammar was extended with the *permcop*y nonterminal to allow existing-practice forms of symbolic modes like *o=u-g* (i.e., set the “other” permissions to the permissions of “owner” minus the permissions of “group”).

4.8 chown — Change file ownership

4.8.1 Synopsis

`chown [-R] owner[:group] file ...`

4.8.2 Description

The `chown` utility shall set the user ID of the file named by each *file* operand to the user ID specified by the *owner* operand.

For each *file* operand, it shall perform actions equivalent to the POSIX.1 {8} `chown()` function, called with the following arguments:

- (1) The *file* operand shall be used as the *path* argument.
- (2) The user ID indicated by the *owner* portion of the first operand shall be used as the *owner* argument.
- (3) If the *group* portion of the first operand is given, the group ID indicated by it shall be used as the *group* argument; otherwise, the group ID of the file shall be used as the *group* argument.

4.8.3 Options

The `chown` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following option shall be supported by the implementation:

- | | |
|-----------|--|
| <p>−R</p> | <p>Recursively change file user IDs, and if the <i>group</i> operand is specified, group IDs. For each <i>file</i> operand that names a directory, <code>chown</code> changes the user and group ID of the directory and all files in the file hierarchy below it.</p> |
|-----------|--|

4.8.4 Operands

The following operands shall be supported by the implementation:

- | | |
|-----------------------------|---|
| <p><i>owner[:group]</i></p> | <p>A user ID and optional group ID to be assigned to file. The <i>owner</i> portion of this operand shall be a user name from the user database or a numeric user ID. Either specifies a user ID to be given to each file named by one of the <i>file</i> operands. If a numeric <i>owner</i> operand exists in the user database as a user name, the user ID number associated with that user name is used as the user ID. Similarly, if the <i>group</i> portion of this operand is present, it shall be a group name from the group database or a numeric group ID. Either specifies a group ID to be given to each file. If a numeric</p> |
|-----------------------------|---|

2840 group operand exists in the group database as a group name, the
 2841 group ID number associated with that group name shall be used
 2842 as the group ID.

2843 *file* A pathname of a file whose user ID is to be modified.

2844 **4.8.5 External Influences**

2845 **4.8.5.1 Standard Input**

2846 None.

2847 **4.8.5.2 Input Files**

2848 None.

2849 **4.8.5.3 Environment Variables**

2850 The following environment variables shall affect the execution of `chown`:

2851 **LANG** This variable shall determine the locale to use for the
 2852 locale categories when both **LC_ALL** and the correspond-
 2853 ing environment variable (beginning with **LC_**) do not
 2854 specify a locale. See 2.6.

2855 **LC_ALL** This variable shall determine the locale to be used to over-
 2856 ride any values for locale categories specified by the set-
 2857 tings of **LANG** or any environment variables beginning
 2858 with **LC_**.

2859 **LC_CTYPE** This variable shall determine the locale for the interpreta-
 2860 tion of sequences of bytes of text data as characters (e.g.,
 2861 single- versus multibyte characters in arguments).

2862 **LC_MESSAGES** This variable shall determine the language in which mes-
 2863 sages should be written.

2864 **4.8.5.4 Asynchronous Events**

2865 Default.

2866 **4.8.6 External Effects**

2867 **4.8.6.1 Standard Output**

2868 None.

4.8.6.2 Standard Error

Used only for diagnostic messages.

4.8.6.3 Output Files

None.

4.8.7 Extended Description

None.

4.8.8 Exit Status

The `chown` utility shall exit with one of the following values:

- 0 The utility executed successfully and all requested changes were made.
- >0 An error occurred.

4.8.9 Consequences of Errors

If, when invoked with the `-R` option, `chown` attempts but fails to change the user ID and/or, if the *group* operand is specified, group ID, of a particular file in a specified file hierarchy, it shall continue to process the remaining files in the hierarchy.

If `chown` cannot read or search a directory within a hierarchy, it shall continue to process the other parts of the hierarchy that are accessible.

4.8.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it can overflow the range of valid exit status values. These are masked by specifying only 0 and >0 as exit values.

The functionality of `chown` is described substantially through references to functions in POSIX.1 {8}. In this way, there is no duplication of effort required for describing the interactions of permissions, multiple groups, etc.

For implementations on which symbolic links are supported, actual use of the `chown()` function to implement this utility might not be the appropriate, depending on the implementation.

History of Decisions Made

The 4.3BSD method of specifying both owner and group was included in this standard because:

- (1) There are cases where the desired end condition could not be achieved using the `chgrp` and `chown` (that only changed the user ID) utilities. [If the current owner is not a member of the desired group and the desired owner is not a member of the current group, the `chown()` function could fail unless both owner and group are changed at the same time.]
- (2) Even if they could be changed independently, in cases where both are being changed, there is a 100 percent performance penalty caused by being forced to invoke both utilities.

The BSD syntax `user[.group]` was changed to `user[:group]` in POSIX.2 because the period is a valid character in login names (as specified by POSIX.1 {8}, login names consist of characters in the portable filename character set). The colon character was chosen as the replacement for the period character because it would never be allowed as a character in a user name or group name on traditional implementations.

The `-R` option is considered by some observers as an undesirable departure from the traditional UNIX system tools approach; since a tool, `find`, already exists to recurse over directories, there was felt to be no good reason to require other tools to have to duplicate that functionality. However, the `-R` option was deemed an important user convenience, is far more efficient than forking a separate process for each element of the directory hierarchy, and is in widespread historical use.

4.9 cksum — Write file checksums and sizes

4.9.1 Synopsis

`cksum` [*file* ...]

4.9.2 Description

The `cksum` utility shall calculate and write to standard output a cyclic redundancy check (CRC) for each input file, and also write to standard output the number of octets in each file. The CRC used is based on the polynomial used for CRC error checking in the networking standard ISO 8802-3 [B7].

The CRC checksum shall be obtained in the following way:

The encoding is defined by the generating polynomial:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Mathematically, the CRC value corresponding to a given file shall be defined by the following procedure:

- (1) The n bits to be evaluated are considered to be the coefficients of a mod 2 polynomial $M(x)$ of degree $n-1$. These n bits are the bits from the file, with the most significant bit being the most significant bit of the first octet of the file and the last bit being the least significant bit of the last octet, padded with zero bits (if necessary) to achieve an integral number of octets, followed by one or more octets representing the length of the file as a binary value, least significant octet first. The smallest number of octets capable of representing this integer shall be used.
- (2) $M(x)$ is multiplied by x^{32} (i.e., shifted left 32 bits) and divided by $G(x)$ using mod 2 division, producing a remainder $R(x)$ of degree ≤ 31 .
- (3) The coefficients of $R(x)$ are considered to be a 32-bit sequence.
- (4) The bit sequence is complemented and the result is the CRC.

4.9.3 Options

None.

4.9.4 Operands

The following operand shall be supported by the implementation:

file A pathname of a file to be checked. If no *file* operands are specified, the standard input is used.

4.9.5 External Influences

4.9.5.1 Standard Input

The standard input is used only if no *file* operands are specified. See Input Files.

4.9.5.2 Input Files

The input files can be any file type.

4.9.5.3 Environment Variables

The following environment variables shall affect the execution of `cksum`:

LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).
LC_MESSAGES	This variable shall determine the language in which messages should be written.

4.9.5.4 Asynchronous Events

Default.

4.9.6 External Effects

4.9.6.1 Standard Output

For each file processed successfully, the `cksum` utility shall write in the following format: 2

"%u %d %s\n", *<checksum>*, *<# of octets>*, *<pathname>* 2

If no *file* operand was specified, the *pathname* and its leading space shall be omitted.

4.9.6.2 Standard Error

Used only for diagnostic messages.

4.9.6.3 Output Files

None.

4.9.7 Extended Description

None.

4.9.8 Exit Status

The `cksum` utility shall exit with one of the following values:

0 All files were processed successfully.

>0 An error occurred.

4.9.9 Consequences of Errors

Default.

4.9.10 Rationale. *(This subclause is not a part of P1003.2)***Examples, Usage**

The `cksum` utility is typically used to quickly compare a suspect file against a trusted version of the same. However, no claims are made by POSIX.2 that this comparison is cryptographically secure; the historical `sum` utility from which `cksum` was inspired has traditionally been used mainly to ensure that files transmitted over noisy media arrive intact. The chances of a damaged file producing the same CRC as the original are astronomically small; deliberate deception is difficult, but probably not impossible.

Although input files to `cksum` can be any type, the results need not be what would be expected on character special device files or on file types not described by POSIX.1 {8}. Since POSIX.2 does not specify the block size used when doing input, checksums of character special files need not process all of the data in those files.

The algorithm is expressed in terms of a bitstream divided into octets. If a file is transmitted between two systems and undergoes any data transformation (such as moving 8-bit characters into 9-bit bytes or changing “little Endian” byte ordering to “big Endian”), identical CRC values cannot be expected. Implementations performing such transformations may extend `cksum` to handle such situations.

The following C-language program can be used as a model to describe the algorithm. It assumes that a `char` is one octet. It also assumes that the entire file is

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.


```

3013 available for one pass through the function. This was done for simplicity in 2
3014 demonstrating the algorithm, rather than as an implementation model. 2
3015 static unsigned long crctab[] = { 2
3016 0x0, 2
3017 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f,
3018 0xe963a535, 0x9e6495a3, 0x0edb8832, 0x79dcb8a4, 0xe0d5e91e,
3019 0x97d2d988, 0x09b64c2b, 0x7eb17cbd, 0xe7b82d07, 0x90bffd91,
3020 0x1db71064, 0x6ab020f2, 0xf3b97148, 0x84be41de, 0x1dad47d,
3021 0x6ddde4eb, 0xf4d4b551, 0x83d385c7, 0x136c9856, 0x646ba8c0,
3022 0xfd62f97a, 0x8a65c9ec, 0x14015c4f, 0x63066cd9, 0xfa0f3d63,
3023 0x8d080df5, 0x3b6e20c8, 0x4c69105e, 0xd56041e4, 0xa2677172,
3024 0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b, 0x35b5a8fa,
3025 0x42b2986c, 0xdbbbc9d6, 0xacbcf940, 0x32d86ce3, 0x45df5c75,
3026 0xdcd60dcf, 0xabd13d59, 0x26d930ac, 0x51de003a, 0xc8d75180,
3027 0xbfd06116, 0x21b4f4b5, 0x56b3c423, 0xcfba9599, 0xb8bda50f,
3028 0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924, 0x2f6f7c87,
3029 0x58684c11, 0xc1611dab, 0xb6662d3d, 0x76dc4190, 0x01db7106,
3030 0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f, 0x9fbfe4a5,
3031 0xe8b8d433, 0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818,
3032 0x7f6a0dbb, 0x086d3d2d, 0x91646c97, 0xe6635c01, 0x6b6b51f4,
3033 0x1c6c6162, 0x856530d8, 0xf262004e, 0x6c0695ed, 0x1b01a57b,
3034 0x8208f4c1, 0xf50fc457, 0x65b0d9c6, 0x12b7e950, 0x8bbeb8ea,
3035 0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3, 0xfbd44c65,
3036 0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2, 0x4adfa541,
3037 0x3dd895d7, 0xa4d1c46d, 0xd3d6f4fb, 0x4369e96a, 0x346ed9fc,
3038 0xad678846, 0xda60b8d0, 0x44042d73, 0x33031de5, 0xaa0a4c5f,
3039 0xdd0d7cc9, 0x5005713c, 0x270241aa, 0xbe0b1010, 0xc90c2086,
3040 0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f, 0x5edef90e,
3041 0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17, 0x2eb40d81,
3042 0xb7bd5c3b, 0xc0ba6cad, 0xedb88320, 0x9abfb3b6, 0x03b6e20c,
3043 0x74b1d29a, 0xead54739, 0x9dd277af, 0x04db2615, 0x73dc1683,
3044 0xe3630b12, 0x94643b84, 0x0d6d6a3e, 0x7a6a5aa8, 0xe40ecf0b,
3045 0x9309fff9, 0x0a00ae27, 0x7d079eb1, 0xf00f9344, 0x8708a3d2,
3046 0x1e01f268, 0x6906c2fe, 0xf762575d, 0x806567cb, 0x196c3671,
3047 0x6e6b06e7, 0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc,
3048 0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5, 0xd6d6a3e8,
3049 0xa1d1937e, 0x38d8c2c4, 0x4fdff252, 0xd1bb67f1, 0xa6bc5767,
3050 0x3fb506dd, 0x48b2364b, 0xd80d2bda, 0xaf0alb4c, 0x36034af6,
3051 0x41047a60, 0xdf60efc3, 0xa867df55, 0x316e8eef, 0x4669be79,
3052 0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236, 0xcc0c7795,
3053 0xbb0b4703, 0x220216b9, 0x5505262f, 0xc5ba3bbe, 0xb2bd0b28,
3054 0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7, 0xb5d0cf31, 0x2cd99e8b,
3055 0x5bdeae1d, 0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a,
3056 0x9c0906a9, 0xeb0e363f, 0x72076785, 0x05005713, 0x95bf4a82,
3057 0xe2b87a14, 0x7bb12bae, 0x0cb61b38, 0x92d28e9b, 0xe5d5be0d,
3058 0x7cdcefb7, 0x0bdbdf21, 0x86d3d2d4, 0xf1d4e242, 0x68ddb3f8,
3059 0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1, 0x18b74777,
3060 0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c, 0x8f659eff,
3061 0xf862ae69, 0x616bffd3, 0x166ccf45, 0xa00ae278, 0xd70dd2ee,
3062 0x4e048354, 0x3903b3c2, 0xa7672661, 0xd06016f7, 0x4969474d,
3063 0x3e6e77db, 0xaed16a4a, 0xd9d65adc, 0x40df0b66, 0x37d83bf0,

```

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

```

3064 0xa9bcae53, 0xdebb9ec5, 0x47b2cf7f, 0x30b5ffe9, 0xbdbdf21c,
3065 0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605, 0xcdd70693,
3066 0x54de5729, 0x23d967bf, 0xb3667a2e, 0xc4614ab8, 0x5d681b02,
3067 0x2a6f2b94, 0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d
3068 };

3069 unsigned long memcrc(const unsigned char *b, size_t n)           2
3070 {                                                                 1
3071 /*      Input arguments:                                         1
3072 *      const char*      b == byte sequence to checksum          1
3073 *      size_t           n == length of sequence                  1
3074 */                                                                1

3075     register unsigned int    i, c, s = 0;                        2

3076     for (i = n; i > 0; --i) {                                     2
3077         c = (unsigned int)(*b++);                                 2
3078         s = (s << 8) ^ crctab[(s >> 24) ^ c];                    2
3079     }                                                             2

3080     /* extend with the length of the string */                   2
3081     while (n != 0) {                                              2
3082         c = n & 0377;                                             2
3083         n >>= 8;                                                  2
3084         s = (s << 8) ^ crctab[(s >> 24) ^ c];                    2
3085     }                                                             2

3086     return ~s;                                                    2
3087 }

```

3088 History of Decisions Made

3089 The historical practice of writing the number of “blocks” has been removed in
 3090 favor of writing the number of octets since the latter is not only more useful, but 2
 3091 historical implementations have not been consistent in defining what a “block”
 3092 meant. Octets are used instead of bytes because bytes can differ in size between
 3093 systems.

3094 The algorithm used was selected to increase the robustness of the utility’s opera-
 3095 tion. Neither the System V nor BSD `sum` algorithm was selected. Since each of
 3096 these was different and each was the default behavior on those systems, no realis-
 3097 tic compromise was available if either were selected—some set of historical appli-
 3098 cations would break. Therefore, the name was changed to `cksum`. Although the
 3099 historical `sum` commands will probably continue to be provided for many years to
 3100 come, programs designed for portability across systems should use the new name.

3101 The algorithm selected is based on that used by the Ethernet standard for the
 3102 Frame Check Sequence Field. The algorithm used does not match the technical
 3103 definition of a *checksum*; the term is used for historical reasons. The length of the 2
 3104 file is included in the CRC calculation because this parallels Ethernet’s inclusion 2
 3105 of a length field in its CRC, but also because it guards against inadvertent colli- 2
 3106 sions between files that begin with different series of zero octets. The chance that 2

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

two different files will produce identical CRCs is much greater when their lengths are not considered. Keeping the length and the checksum of the file itself separate would yield a slightly more robust algorithm, but historical usage has always been that a single number (the checksum as printed) represents the signature of the file. It was decided that historical usage was the more important consideration.

Earlier drafts contained modifications to the Ethernet algorithm that involved extracting table values whenever an intermediate result became zero. This was demonstrated to be less robust than the current method and mathematically difficult to describe or justify.

Editor's Note: The following bibliographic references will be cleaned up before the standard is completed.

The calculation used is identical to that given in pseudo-code on page 1011 of *Communications of the ACM*, August, 1988 in the article "Computation of Cyclic Redundancy Checks Via Table Lookup" by Dilip V. Sarwate. The pseudo-code rendition is:

```

X <- 0; Y <- 0;
for i <- m -1 step -1 until 0 do
    begin
        T <- X(1) ^ A[i];
        X(1) <- X(0); X(0) <- Y(1); Y(1) <- Y(0); Y(0) <- 0;
        comment: f[T] and f'[T] denote the T-th words in the
                  table f and f' ;
        X <- X ^ f[T]; Y <- Y ^ f'[T];
    end

```

The pseudo-code is reproduced exactly as given; however, note that in cksum's case, A[i] represents a byte of the file, the words X and Y are treated as a single 32-bit value, and the tables f and f' are a single table containing 32-bit values.

The article also discusses generating the table(s).

Other sources consulted about CRC's:

"A Tutorial on CRC Computations," Ramabadran and Gaitonde, *IEEE Micro*, p. 62, August 1988;

Computer Networks, Andrew Tanenbaum, Prentice-Hall, Inc.

3141 **4.10 cmp — Compare two files**

3142 **4.10.1 Synopsis**

3143 `cmp [-l | -s] file1 file2`

3144 **4.10.2 Description**

3145 The `cmp` utility shall compare two files. The `cmp` utility shall write no output if
 3146 the files are the same. Under default options, if they differ, it shall write to stan-
 3147 dard output the byte and line number at which the first difference occurred.
 3148 Bytes and lines shall be numbered beginning with 1.

3149 **4.10.3 Options**

3150 The `cmp` utility shall conform to the utility argument syntax guidelines described
 3151 in 2.10.2.

3152 The following options shall be supported by the implementation:

3153	-l	(Lowercase ell.) Write the byte number (decimal) and the differ-
3154		ing bytes (octal) for each difference.
3155	-s	Write nothing for differing files; return exit status only.

3156 **4.10.4 Operands**

3157 The following operands shall be supported by the implementation:

3158	<i>file1</i>	A pathname of the first file to be compared. If <i>file1</i> is –, the stan-
3159		dard input shall be used.
3160	<i>file2</i>	A pathname of the second file to be compared. If <i>file2</i> is –, the
3161		standard input shall be used.

3162 If both *file1* and *file2* refer to standard input or refer to the same FIFO special,
 3163 block special, or character special file, the results are undefined.

3164 **4.10.5 External Influences**

3165 **4.10.5.1 Standard Input**

3166 The standard input shall be used only if the *file1* or *file2* operand refers to stan-
 3167 dard input. See Input Files.

4.10.5.2 Input Files

The input files can be any file type.

4.10.5.3 Environment Variables

The following environment variables shall affect the execution of `cmp`:

LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).
LC_MESSAGES	This variable shall determine the language in which messages should be written.

4.10.5.4 Asynchronous Events

Default.

4.10.6 External Effects

4.10.6.1 Standard Output

In the POSIX Locale, results of the comparison shall be written to standard output. When no options are used, the format shall be:

```
"%s %s differ: char %d, line %d\n", file1, file2, <byte number>,
<line number>
```

When the `-l` option is used, the format is:

```
"%d %o %o\n", <byte number>, <differing byte>, <differing byte>
```

for each byte that differs. The first *<differing byte>* number is from *file1* while the second is from *file2*. In both cases, *<byte number>* shall be relative to the beginning of the file, beginning with 1.

The *<additional info>* field shall either be null or a string that starts with a *<blank>* and contains no *<newline>* characters.

No output shall be written to standard output when the `-s` option is used.

3201 **4.10.6.2 Standard Error**

3202 Used only for diagnostic messages. If *file1* and *file2* are identical for the entire 2
 3203 length of the shorter file, in the POSIX Locale the following diagnostic message 2
 3204 shall be written, unless the `-s` option is specified. 2

3205 "cmp: EOF on %s%s\n", *<name of shorter file>*, *<additional info>* 1

3206 **4.10.6.3 Output Files**

3207 None.

3208 **4.10.7 Extended Description**

3209 None.

3210 **4.10.8 Exit Status**

3211 The `cmp` utility shall exit with one of the following values:

- 3212 0 The files are identical.
- 3213 1 The files are different; this includes the case where one file is identical
 3214 to the first part of the other.
- 3215 >1 An error occurred.

3216 **4.10.9 Consequences of Errors**

3217 Default.

3218 **4.10.10 Rationale.** (*This subclause is not a part of P1003.2*)

3219 **Examples, Usage**

3220 The global language in Section 2 indicates that using two mutually-exclusive
 3221 options together produces unspecified results. Some System V implementations
 3222 consider the option usage:

3223 `cmp -l -s ...`

3224 to be an error. They also treat:

3225 `cmp -s -l ...`

3226 as if no options were specified. Both of these behaviors are considered bugs, but
 3227 are allowed.

3228 Although input files to `cmp` can be any type, the results might not be what would
 3229 be expected on character special device files or on file types not described by
 3230 POSIX.1 {8}. Since POSIX.2 does not specify the block size used when doing input,

3231 comparisons of character special files need not compare all of the data in those
 3232 files.

3233 The word `char` in the standard output format comes from historical usage, even 1
 3234 though it is actually a byte number. When `cmp` is supported in other locales, 1
 3235 implementations are encouraged to use the word `byte` or its equivalent in 1
 3236 another language. Users should not interpret this difference to indicate that the 1
 3237 functionality of the utility changed between locales. 1

3238 **History of Decisions Made**

3239 Some systems report on the number of lines in the identical-but-shorter file case. 1
 3240 This is allowed by the inclusion of the *<additional info>* fields in the output for- 1
 3241 mat. The restriction on having a leading *<blank>* and no *<newline>*s is to make 1
 3242 parsing for the file name easier. It is recognized that some file names containing 1
 3243 white-space characters will make parsing difficult anyway, but the restriction 1
 3244 does aid programs used on systems where the names are predominantly well 1
 3245 behaved. 1

3246 4.11 `comm` — Select or reject lines common to two files

3247 4.11.1 Synopsis

3248 `comm [-123] file1 file2`

3249 4.11.2 Description

3250 The `comm` utility shall read *file1* and *file2*, which should be ordered in the current
 3251 collating sequence, and produce three text columns as output: lines only in *file1*;
 3252 lines only in *file2*; and lines in both files.

3253 If the lines in both files are not ordered according to the collating sequence of the
 3254 current locale, the results are unspecified.

3255 4.11.3 Options

3256 The `comm` utility shall conform to the utility argument syntax guidelines
 3257 described in 2.10.2.

3258 The following options shall be supported by the implementation:

3259	-1	Suppress the output column of lines unique to <i>file1</i> .	1
3260	-2	Suppress the output column of lines unique to <i>file2</i> .	1

3261 -3 Suppress the output column of lines duplicated in *file1* and *file2*. 1

3262 **4.11.4 Operands**

3263 The following operands shall be supported by the implementation:

3264 *file1* A pathname of the first file to be compared. If *file1* is -, the stan-
3265 dard input is used.

3266 *file2* A pathname of the second file to be compared. If *file2* is -, the
3267 standard input is used.

3268 If both *file1* and *file2* refer to standard input or to the same FIFO special, block
3269 special, or character special file, the results are undefined.

3270 **4.11.5 External Influences**

3271 **4.11.5.1 Standard Input**

3272 The standard input shall be used only if one of the *file1* or *file2* operands refers to
3273 standard input. See Input Files.

3274 **4.11.5.2 Input Files**

3275 The input files shall be text files.

3276 **4.11.5.3 Environment Variables**

3277 The following environment variables shall affect the execution of `comm`:

3278 **LANG** This variable shall determine the locale to use for the
3279 locale categories when both **LC_ALL** and the correspond-
3280 ing environment variable (beginning with **LC_**) do not
3281 specify a locale. See 2.6.

3282 **LC_ALL** This variable shall determine the locale to be used to over-
3283 ride any values for locale categories specified by the set-
3284 tings of **LANG** or any environment variables beginning
3285 with **LC_**.

3286 **LC_CTYPE** This variable shall determine the locale for the interpreta-
3287 tion of sequences of bytes of text data as characters (e.g.,
3288 single- versus multibyte characters in arguments and
3289 input files).

3290 **LC_COLLATE** This variable shall determine the locale for the collating
3291 sequence `comm` expects to have been used when the input
3292 files were sorted.

3293 **LC_MESSAGES** This variable shall determine the language in which mes-
 3294 sages should be written.

3295 **4.11.5.4 Asynchronous Events**

3296 Default.

3297 **4.11.6 External Effects**

3298 **4.11.6.1 Standard Output**

3299 The `comm` utility shall produce output depending on the options selected. If the
 3300 `-1`, `-2`, and `-3` options are all selected, `comm` shall write nothing to standard out-
 3301 put.

3302 If the `-1` option is not selected, lines contained only in *file1* shall be written using
 3303 the format:

3304 "`%s\n`", *<line in file1>*

3305 If the `-2` option is not selected, lines contained only in *file2* shall be written using
 3306 the format:

3307 "`%s%s\n`", *<lead>*, *<line in file2>*

3308 where the string *<lead>* is:

3309 <tab> if the `-1` option is not selected, or

3310 null string if the `-1` option is selected.

3311 If the `-3` option is not selected, lines contained in both files shall be written using
 3312 the format:

3313 "`%s%s\n`", *<lead>*, *<line in both>*

3314 where the string *<lead>* is:

3315 <tab><tab> if neither the `-1` nor the `-2` option is selected, or

3316 <tab> if exactly one of the `-1` and `-2` options is selected, or

3317 null string if both the `-1` and `-2` options are selected.

3318 If the input files were ordered according to the collating sequence of the current
 3319 locale, the lines written shall be in the collating sequence of the original lines.

3320 **4.11.6.2 Standard Error**

3321 Used only for diagnostic messages.

3322 **4.11.6.3 Output Files**

3323 None.

3324 **4.11.7 Extended Description**

3325 None.

3326 **4.11.8 Exit Status**3327 The `comm` utility shall exit with one of the following values:

3328 0 All input files were successfully output as specified.

3329 >0 An error occurred.

3330 **4.11.9 Consequences of Errors**

3331 Default.

3332 **4.11.10 Rationale.** *(This subclause is not a part of P1003.2)*3333 **Examples, Usage**3334 If the input files are not properly presorted, the output of `comm` might not be use-
3335 ful.3336 If a file named `posix.2` contains a sorted list of the utilities in this standard, a
3337 file named `xpg3` contains a sorted list of the utilities specified in X/Open Portabil-
3338 ity Guide Issue 3, and a file named `svid89` contains a sorted list of the utilities in
3339 the System V Interface Definition Third Edition:3340 `comm -23 posix.2 xpg3 | comm -23 - svid89`3341 would print a list of utilities in this standard not specified by either of the other
3342 documents,3343 `comm -12 posix.2 xpg3 | comm -12 - svid89`

3344 would print a list of utilities specified by all three documents, and

3345 `comm -12 xpg3 svid89 | comm -23 - posix.2`3346 would print a list of utilities specified by both `XPG3` and `SVID`, but not specified in
3347 this standard.3348 **History of Decisions Made**

3349 None.

3350 **4.12 command — Execute a simple command**

3351 **4.12.1 Synopsis**

3352 `command [-p] command_name [argument ...]`

3353 **4.12.2 Description**

3354 The `command` utility shall cause the shell to treat the arguments as a simple com- 1
 3355 mand, suppressing the shell function lookup that is described in 3.9.1.1 item 1
 3356 (1)(b).

3357 If the *command_name* is the same as the name of one of the special built-in utili-
 3358 ties, the special properties in the enumerated list at the beginning of 3.14 shall
 3359 not occur. In every other respect, if *command_name* is not the name of a function,
 3360 the effect of `command` shall be the same as omitting `command`.

3361 **4.12.3 Options**

3362 The `command` utility shall conform to the utility argument syntax guidelines
 3363 described in 2.10.2.

3364 The following option shall be supported by the implementation:

3365	-p	Perform the command search using a default value for PATH that
3366		is guaranteed to find all of the standard utilities.

3367 **4.12.4 Operands**

3368 The following operands shall be supported by the implementation:

3369	<i>argument</i>	One of the strings treated as an argument to <i>command_name</i> .
3370	<i>command_name</i>	The name of a utility or a special built-in utility.
3371		

3372 **4.12.5 External Influences**

3373 **4.12.5.1 Standard Input**

3374 None.

3375 **4.12.5.2 Input Files**

3376 None.

4.12.5.3 Environment Variables

The following environment variables shall affect the execution of `command`:

LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters).
LC_MESSAGES	This variable shall determine the language in which messages should be written.
PATH	This variable shall determine the search path used during the command search described in 3.9.1.1, except as described under the <code>-p</code> option.

4.12.5.4 Asynchronous Events

Default.

4.12.6 External Effects

4.12.6.1 Standard Output

None.

4.12.6.2 Standard Error

Used only for diagnostic messages.

4.12.6.3 Output Files

None.

4.12.7 Extended Description

None.

4.12.8 Exit Status

The `command` utility shall exit with one of the following values:

126 The utility specified by *command_name* was found but could not be invoked. 1

127 An error occurred in the `command` utility or the utility specified by *command_name* could not be found. 1

Otherwise, the exit status of `command` shall be that of the simple command specified by the arguments to `command`.

4.12.9 Consequences of Errors

Default.

4.12.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The order for command search in POSIX.2 allows functions to override regular built-ins and path searches. This utility is necessary to allow functions that have the same name as a utility to call the utility (instead of a recursive call to the function).

The system default path is available using `getconf`; however, since `getconf` may need to have the **PATH** set up before it can be called itself, the following can be used:

```
command -p getconf _CS_PATH
```

Since `command` appears in Table 2-2, it will always be found prior to the **PATH** search.

There is nothing in the description of `command` that implies the command line is parsed any differently than for any other simple command. For example,

```
command a | b ; c
```

is not parsed in any special way that causes `|` or `;` to be treated other than a pipe operator or semicolon or that prevents function lookup on `b` or `c`.

Examples: Make a version of `cd` that always prints out the new working directory exactly once:

```

3435     cd() {
3436         command cd "$@" >/dev/null
3437         pwd
3438     }

```

3439 Start off a “secure shell script” in which the script avoids being spoofed by its
 3440 parent:

```

3441     IFS= '
3442     '
3443     #         The preceding value should be <space><tab><newline>.
3444     #         Set IFS to its default value.
3445     \unset -f command
3446     #         Ensure command is not a user function.
3447     #         Note that unset is escaped to prevent an alias being used
3448     #         for unset on implementations that support aliases.
3449     PATH="$(\command -p getconf _CS_PATH):$PATH"
3450     #         Put on a reliable PATH prefix.
3451     #         Now, unset all utility names that will be used (or
3452     #         invoke them with \command each time).
3453     #         ...

```

1

3454 At this point, given correct permissions on the directories called by **PATH**, the
 3455 script has the ability to ensure that any utility it calls is the intended one. It is
 3456 being very cautious because it assumes that implementation extensions may be
 3457 present that would allow user aliases and/or functions to exist when it is invoked;
 3458 neither capability is specified by POSIX.2, but neither is prohibited as an exten-
 3459 sion. For example, the proposed UPE supplement to POSIX.2 introduces a **ENV**
 3460 variable that precedes the invocation of the script with a user startup script.
 3461 Such a script could have used the aliasing facility from the UPE or the functions
 3462 in POSIX.2 to spoof the application.

3463 The `command`, `env`, `nohup`, and `xargs` utilities have been specified to use exit
 3464 code 127 if an error occurs so that applications can distinguish “failure to find a
 3465 utility” from “invoked utility exited with an error indication.” The value 127 was
 3466 chosen because it is not commonly used for other meanings; most utilities use
 3467 small values for “normal error conditions” and the values above 128 can be con-
 3468 fused with termination due to receipt of a signal. The value 126 was chosen in a
 3469 similar manner to indicate that the utility could be found, but not invoked. Some
 3470 scripts produce meaningful error messages differentiating the 126 and 127 cases.
 3471 The distinction between exit codes 126 and 127 is based on KornShell practice
 3472 that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses
 3473 126 when any attempt to *exec* the utility fails for any other reason.

1

1

1

1

1

2

2

2

3474 History of Decisions Made

3475 The `command` utility is somewhat similar to the Eighth Edition builtin com-
 3476 mand, but since `command` also goes to the file system to search for utilities, the
 3477 name builtin would not be intuitive.

The `command` utility will most likely be provided as a regular built-in. In an earlier draft, it was a special built-in. This was changed for the following reasons:

- The removal of exportable functions made the special precedence of a special built-in unnecessary.

- A special built-in has special properties (see the enumerated list at the beginning of 3.14) that were inappropriate for invoking other utilities. For example, two commands such as

```
date > unwritable-file
```

```
command date > unwritable-file
```

would have entirely different results; in a noninteractive script, the former would continue to execute the next command, the latter would abort. Introducing this semantic difference along with suppressing functions was seen to be nonintuitive.

- There are some advantages of suppressing the special characteristics of special built-ins on occasion. For example:

```
command exec > unwritable-file
```

will not cause a noninteractive script to abort, so that the output status can be checked by the script.

An earlier draft presented a larger number of options. Most were removed because they were not useful to real portable applications, given the new `command` search order.

The `-p` option is present because it is useful to be able to ensure a safe path search that will find all the POSIX.2 standard utilities. This search might not be identical to the one that occurs through one of the POSIX.1 {8} `exec` functions when **PATH** is unset, as explained in 2.6.1. At the very least, this feature is required to allow the script to access the correct version of `getconf` so that the value of the default path can be accurately retrieved.

4.13 `cp` — Copy files

4.13.1 Synopsis

```

cp [-fip] source_file target_file      2
cp [-fip] source_file ... target      2
cp -R [-fip] source_file ... target   2
cp -r [-fip] source_file ... target   2

```

4.13.2 Description

The first synopsis form is denoted by two operands, neither of which are existing files of type directory. The `cp` utility shall copy the contents of *source_file* to the destination path named by *target_file*.

The second synopsis form is denoted by two or more operands where the `-R` or `-r` options are not specified and the first synopsis form is not applicable. It shall be an error if any *source_file* is a file of type directory, if *target* does not exist, or if *target* is a file of a type defined by POSIX.1 {8}, but is not a file of type directory. The `cp` utility shall copy the contents of each *source_file* to the destination path named by the concatenation of *target*, a slash character, and the last component of *source_file*.

The third and fourth synopsis forms are denoted by two or more operands where the `-R` or `-r` options are specified. The `cp` utility shall copy each file in the file hierarchy rooted in each *source_file* to a destination path named as follows.

If *target* exists and is a file of type directory, the name of the corresponding destination path for each file in the file hierarchy shall be the concatenation of *target*, a slash character, and the pathname of the file relative to the directory containing *source_file*.

If *target* does not exist, and two operands are specified, the name of the corresponding destination path for *source_file* shall be *target*; the name of the corresponding destination path for all other files in the file hierarchy shall be the concatenation of *target*, a slash character, and the pathname of the file relative to *source_file*.

It shall be an error if *target* does not exist and more than two operands are specified, or if *target* exists and is a file of a type defined by POSIX.1 {8}, but is not a file of type directory.

In the following description, *source_file* refers to the file that is being copied, whether specified as an operand or a file in a file hierarchy rooted in a *source_file* operand. The term *dest_file* refers to the file named by the destination path.

For each *source_file*, the following steps shall be taken:

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

- 3541 (1) If *source_file* references the same file as *dest_file*, *cp* may write a diagnos-
 3542 tic message to standard error; it shall do nothing more with *source_file* 1
 3543 and shall go on to any remaining files. 1
- 3544 (2) If *source_file* is of type directory, the following steps shall be taken:
- 3545 (a) If neither the *-R* or *-r* options were specified, *cp* shall write a diag-
 3546 nostic message to standard error, do nothing more with *source_file*,
 3547 and go on to any remaining files.
- 3548 (b) If *source_file* was not specified as an operand and *source_file* is dot
 3549 or dot-dot, *cp* shall do nothing more with *source_file* and go on to
 3550 any remaining files.
- 3551 (c) If *dest_file* exists and it is a file type not specified by POSIX.1 {8}, the
 3552 behavior is implementation defined.
- 3553 (d) If *dest_file* exists and it is not of type directory, *cp* shall write a
 3554 diagnostic message to standard error, do nothing more with
 3555 *source_file* or any files below *source_file* in the file hierarchy, and go
 3556 on to any remaining files.
- 3557 (e) If the directory *dest_file* does not exist, it shall be created with file
 3558 permission bits set to the same value as those of *source_file*,
 3559 modified by the file creation mask of the user if the *-p* option was
 3560 not specified, and then bitwise inclusively ORed with *S_IRWXU*. If
 3561 *dest_file* cannot be created, *cp* shall write a diagnostic message to
 3562 standard error, do nothing more with *source_file*, and go on to any
 3563 remaining files. It is unspecified if *cp* shall attempt to copy files in
 3564 the file hierarchy rooted in *source_file*.
- 3565 (f) The files in the directory *source_file* shall be copied to the directory
 3566 *dest_file*, taking the four steps [(1)-(4)] listed here with the files as
 3567 *source_files*.
- 3568 (g) If *dest_file* was created, its file permission bits shall be changed (if
 3569 necessary) to be the same as those of *source_file*, modified by the file
 3570 creation mask of the user if the *-p* option was not specified.
- 3571 (h) The *cp* utility shall do nothing more with *source_file* and go on to
 3572 any remaining files.
- 3573 (3) If *source_file* is of type regular file, the following steps shall be taken: 1
- 3574 (a) If *dest_file* exists, the following steps are taken:
- 3575 [1] If the *-i* option is in effect, the *cp* utility shall write a prompt
 3576 to the standard error and read a line from the standard input.
 3577 If the response is not affirmative, *cp* shall do nothing more
 3578 with *source_file* and go on to any remaining files.
- 3579 [2] A file descriptor for *dest_file* shall be obtained by performing
 3580 actions equivalent to the POSIX.1 {8} *open()* function call using
 3581 *dest_file* as the *path* argument, and the bitwise inclusive OR of
 3582 *O_WRONLY* and *O_TRUNC* as the *oflag* argument.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

- 3583 [3] If the attempt to obtain a file descriptor fails and the `-f` option 2
 3584 is in effect, `cp` shall attempt to remove the file by performing 2
 3585 actions equivalent to the POSIX.1 {8} `unlink()` function called 2
 3586 using *dest_file* as the *path* argument. If this attempt succeeds, 2
 3587 `cp` shall continue with step (3b). 2
- 3588 (b) If *dest_file* does not exist, a file descriptor shall be obtained by per-
 3589 forming actions equivalent to the POSIX.1 {8} `open()` function called
 3590 using *dest_file* as the *path* argument, and the bitwise inclusive OR
 3591 of `O_WRONLY` and `O_CREAT` as the *oflag* argument. The file per-
 3592 mission bits of *source_file* shall be the *mode* argument.
- 3593 (c) If the attempt to obtain a file descriptor fails, `cp` shall write a diag-
 3594 nostic message to standard error, do nothing more with *source_file*,
 3595 and go on to any remaining files.
- 3596 (d) The contents of *source_file* shall be written to the file descriptor.
 3597 Any write errors shall cause `cp` to write a diagnostic message to
 3598 standard error and continue to step (3)(e).
- 3599 (e) The file descriptor shall be closed.
- 3600 (f) The `cp` utility shall do nothing more with *source_file*. If a write 2
 3601 error occurred in step (3d), it is unspecified if `cp` continues with any 2
 3602 remaining files. If no write error occurred in step (3d), `cp` shall go 2
 3603 on to any remaining files. 2
- 3604 (4) Otherwise, the following steps shall be taken:
- 3605 (a) If the `-r` option was specified, the behavior is implementation 1
 3606 defined. 1
- 3607 (b) If the `-R` option was specified, the following steps shall be taken: 1
- 3608 [1] The *dest_file* shall be created with the same file type as 1
 3609 *source_file*. 1
- 3610 [2] If *source_file* is a file of type FIFO, the file permission bits shall 1
 3611 be the same as those of *source_file*, modified by the file crea-
 3612 tion mask of the user if the `-p` option was not specified. Oth-
 3613 erwise, the permissions, owner ID, and group ID of *dest_file* are
 3614 implementation defined.
- 3615 If this creation fails for any reason, `cp` shall write a diagnostic
 3616 message to standard error, do nothing more with *source_file*,
 3617 and go on to any remaining files.
- 3618 If the implementation provides additional or alternate access control mechanisms
 3619 (see 2.2.2.55), their effect on copies of files is implementation-defined.

4.13.3 Options

The `cp` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

- | | | | |
|--------------------------------------|----|---|-------------|
| 3624
3625
3626 | -f | If a file descriptor for a destination file cannot be obtained, as described in step (3a)[2], attempt to unlink the destination file and proceed. | 2
2
2 |
| 3627
3628
3629 | -i | Write a prompt to standard error before copying to any existing destination file. If the response from the standard input is affirmative, the copy shall be attempted, otherwise not. | |
| 3630
3631 | -p | Duplicate the following characteristics of each source file in the corresponding destination file: | |
| 3632
3633
3634 | | (1) The time of last data modification and time of last access. If this duplication fails for any reason, <code>cp</code> shall write a diagnostic message to standard error. | |
| 3635
3636
3637 | | (2) The user ID and group ID. If this duplication fails for any reason, it is unspecified whether <code>cp</code> writes a diagnostic message to standard error. | |
| 3638
3639
3640
3641 | | (3) The file permission bits and the <code>S_ISUID</code> and <code>S_ISGID</code> bits. Other, implementation-defined, bits may be duplicated as well. If this duplication fails for any reason, <code>cp</code> shall write a diagnostic message to standard error. | |
| 3642
3643
3644
3645
3646 | | If the user ID or the group ID cannot be duplicated, the file permission bits <code>S_ISUID</code> and <code>S_ISGID</code> shall be cleared. If these bits are present in the source file but are not duplicated in the destination file, it is unspecified whether <code>cp</code> writes a diagnostic message to standard error. | |
| 3647
3648
3649 | | The order in which the preceding characteristics are duplicated is unspecified. The <i>dest_file</i> shall not be deleted if these characteristics cannot be preserved. | |
| 3650 | -R | Copy file hierarchies. | |
| 3651
3652 | -r | Copy file hierarchies. The treatment of special files is implementation defined. | 1
1 |

4.13.4 Operands

The following operands shall be supported by the implementation:

3655 *source_file* A pathname of a file to be copied.

3656 *target_file* A pathname of an existing or nonexisting file, used for the output
 3657 when a single file is copied.

3658 *target* A pathname of a directory to contain the copied file(s).

3659 **4.13.5 External Influences**

3660 **4.13.5.1 Standard Input**

3661 Used to read an input line in response to each prompt specified in Standard
 3662 Error. Otherwise, the standard input shall not be used.

3663 **4.13.5.2 Input Files**

3664 The input files specified as operands may be of any file type.

3665 **4.13.5.3 Environment Variables**

3666 The following environment variables shall affect the execution of `cp`:

3667 LANG	This variable shall determine the locale to use for the
3668	locale categories when both LC_ALL and the correspond-
3669	ing environment variable (beginning with LC_) do not
3670	specify a locale. See 2.6.
3671 LC_ALL	This variable shall determine the locale to be used to over-
3672	ride any values for locale categories specified by the set-
3673	tings of LANG or any environment variables beginning
3674	with LC_ .
3675 LC_COLLATE	This variable shall determine the locale for the behavior of
3676	ranges, equivalence classes, and multicharacter collating
3677	elements used in the extended regular expression defined
3678	for the <code>yesexpr</code> locale keyword in the LC_MESSAGES
3679	category.
3680 LC_CTYPE	This variable shall determine the locale for the interpreta-
3681	tion of sequences of bytes of text data as characters (e.g.,
3682	single- versus multibyte characters in arguments) and the
3683	behavior of character classes used in the extended regular
3684	expression defined for the <code>yesexpr</code> locale keyword in the
3685	LC_MESSAGES category.
3686 LC_MESSAGES	This variable shall determine the processing of affirmative
3687	responses and the language in which messages should be
3688	written.

3689 **4.13.5.4 Asynchronous Events**

3690 Default.

3691 **4.13.6 External Effects**

3692 **4.13.6.1 Standard Output**

3693 None.

3694 **4.13.6.2 Standard Error**

3695 A prompt shall be written to standard error under the conditions specified in
3696 4.13.2. The prompt shall contain the destination pathname, but its format is oth-
3697 erwise unspecified. Otherwise, the standard error shall be used only for diagnos-
3698 tic messages.

3699 **4.13.6.3 Output Files**

3700 The output files may be of any type.

3701 **4.13.7 Extended Description**

3702 None.

3703 **4.13.8 Exit Status**

3704 The `cp` utility shall exit with one of the following values:

3705 0 No error occurred.

3706 >0 An error occurred.

3707 **4.13.9 Consequences of Errors**

3708 If `cp` is prematurely terminated by a signal or error, files or file hierarchies may
3709 be only partially copied and files and directories may have incorrect permissions
3710 or access and modification times.

4.13.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

None.

History of Decisions Made

The `-i` option exists on BSD systems, giving applications and users a way to avoid accidentally removing files when copying. Although the 4.3BSD version does not prompt if the standard input is not a terminal, the working group decided that use of `-i` is a request for interaction, so when the destination path exists, the utility takes instructions from whatever responds on standard input.

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified, because implementations may desire more descriptive prompts than those used on historical implementations. Therefore, an application using the `-i` option relies on the system to provide the most suitable dialogue directly with the user, based on the behavior specified.

The `-p` option is historical practice on BSD systems, duplicating the time of last data modification and time of last access. POSIX.2 extends it to preserve the user and group IDs, as well as the file permissions. This requirement has obvious problems in that the directories are almost certainly modified after being copied. This specification requires that the modification times be preserved even so. The statement that the order in which the characteristics are duplicated is unspecified is to permit implementations to provide the maximum amount of security for the user. Implementations should take into account the obvious security issues involved in setting the owner, group, and mode in the wrong order or creating files with an owner, group, or mode different from the final value.

It is unspecified whether `cp` writes diagnostic messages when the user and group IDs cannot be set due to the widespread practice of users using `-p` to duplicate some portion of the file characteristics, indifferent to the duplication of others. Historic implementations only write diagnostic messages on errors other than [EPERM].

The `-r` option is historical practice on BSD and BSD-derived systems, copying file hierarchies as opposed to single files. This functionality is used heavily in existing applications and its loss would significantly decrease consensus. The `-R` option was added as a close synonym to the `-r` option, selected for consistency with all other options in the standard that do recursive directory descent.

The difference between `-R` and `-r` is in the treatment by `cp` of file types other than regular and directory. The original `-r` flag, for historic reasons, does not handle special files any differently than regular files, but always reads the file and copies its contents. This has obvious problems in the presence of special file types, for example character devices, FIFOs, and sockets. The current `cp` utility specification is intended to require that the `-R` option recreate the file hierarchy and that the `-r` option support historical practice. It is anticipated that a future

version of this standard will deprecate the `-r` option, and for that reason, there has been no attempt to fix its behavior with respect to FIFOs or other file types where copying the file is clearly wrong. However, some systems support `-r` with the same abilities as the `-R` defined in POSIX.2. To accommodate them as well as systems that do not, the differences between `-r` and `-R` are implementation defined. Implementations may make them identical.

When a failure occurs during the copying of a file hierarchy, `cp` is required to attempt to copy files that are on the same level in the hierarchy or above the file where the failure occurred. It is unspecified if `cp` shall attempt to copy files below the file where the failure occurred (which cannot succeed in any case).

Permissions, owners, and groups of created special file types have been deliberately left as implementation defined. This is to allow systems to satisfy special requirements (for example, allowing users to create character special devices, but requiring them to be owned by a certain group). In general, it is strongly suggested that the permissions, owner, and group be the same as if the user had run the traditional `mknod`, `ln`, or other utility to create the file. It is also probable that additional privileges will be required to create block, character, or other, implementation-specific, special file types.

Additionally, the `-p` option explicitly requires that all set-user-ID and set-group-ID permissions be discarded if any of the owner or group IDs cannot be set. This is to keep users from unintentionally giving away special privilege when copying programs.

When creating regular files, historical versions of `cp` use the mode of the source file as modified by the file mode creation mask. Other choices would have been to use the mode of the source file unmodified by the creation mask, or to use the same mode as would be given to a new file created by the user, plus the execution bits of the source file, and then modified by the file mode creation mask. In the absence of any strong reason to change historic practice, it was in large part retained.

The one difference is that the set-user-ID and set-group-ID bits are explicitly cleared when files are created. This is to prevent users from creating programs that are set-user-ID/set-group-ID to them when copying files or to make set-user-ID/set-group-ID files accessible to new groups of users. For example, if a file is set-user-ID and the copy has a different group ID than the source, a new group of users have execute permission to a set-user-ID program than did previously. In particular, this is a problem for super-users copying users' trees. A finer granularity of protection could be specified, in that the set-user-ID/set-group-ID bits could be retained under certain conditions even if the owner or group could not be set, based on a determination that no additional privileges were provided to any users. This was not seen as sufficiently useful for the added complexity.

When creating directories, historical versions of `cp` use the mode of the source directory, plus read, write, and search bits for the owner, as modified by the file mode creation mask. This is done so that `cp` can copy trees where the user has read permission, but the owner does not. A side effect is that if the file creation mask denies the owner permissions, `cp` will fail. Also, once the copy is done,

3797 historical versions of `cp` set the permissions on the created directory to be the
3798 same as the source directory, unmodified by the file creation mask.

3799 This behavior has been modified so that `cp` will always be able to create the con-
3800 tents of the directory, regardless of the file creation mask. After the copy is done,
3801 the permissions are set to be the same as the source directory, as modified by the
3802 file creation mask. This latter change from historical behavior is to prevent users
3803 from accidentally creating directories with permissions beyond those they would
3804 normally set and for consistency with the behavior of `cp` in creating files.

3805 It is not a requirement that `cp` detect attempts to copy a file to itself; however,
3806 implementations are strongly encouraged to do so. Historical implementations
3807 have detected the attempt in most cases, which is probably all that is needed.

3808 There are two methods of copying subtrees in this standard. The other method is
3809 described as part of the `pax` utility (see 4.48). Both methods are historical prac-
3810 tice. The `cp` utility provides a simpler, more intuitive interface, while `pax` offers
3811 a finer granularity of control. Each provides additional functionality to the other;
3812 in particular, `pax` maintains the hard-link structure of the hierarchy, while `cp`
3813 does not. It is the intention of the working group that the results be similar
3814 (using appropriate option combinations in both utilities). The results are not
3815 required to be identical; there seemed insufficient gain to applications to balance
3816 the difficulty of implementations having to guarantee that the results would be
3817 exactly identical.

3818 The wording allowing `cp` to copy a directory to implementation-defined file types
3819 not specified by POSIX.1 {8} is provided so that implementations supporting sym-
3820 bolic links are not required to prohibit copying directories to symbolic links.
3821 Other extensions to POSIX.1 {8} file types may need to use this loophole as well.

4.14 cut — Cut out selected fields of each line of a file

4.14.1 Synopsis

cut -b *list* [-n] [*file* ...]

cut -c *list* [*file* ...]

cut -f *list* [-d *delim*] [-s] [*file* ...]

4.14.2 Description

The `cut` utility shall cut out bytes (-b option), characters (-c option), or character-delimited fields (-f option) from each line in one or more files, concatenate them, and write them to standard output.

4.14.3 Options

The `cut` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The option-argument *list* (see options -b, -c, and -f below) shall be a comma-separated list or <blank>-separated list of positive numbers and ranges. Ranges can be in three forms. The first is two positive numbers separated by a hyphen (*low-high*), which represents all fields from the first number to the second number. The second is a positive number preceded by a hyphen (*-high*), which represents all fields from field number 1 to that number. The third is a positive number followed by a hyphen (*low-*), which represents that number to the last field, inclusive. The elements in *list* can be repeated, can overlap, and can be specified in any order.

The following options shall be supported by the implementation:

-b *list* Cut based on a *list* of bytes. Each selected byte shall be output unless the -n option is also specified. It shall not be an error to select bytes not present in the input line.

-c *list* Cut based on a *list* of characters. Each selected character shall be output. It shall not be an error to select characters not present in the input line.

-d *delim* Set the field delimiter to the character *delim*. The default is the <tab> character.

-f *list* Cut based on a *list* of fields, assumed to be separated in the file by a delimiter character (see -d). Each selected field shall be output. Output fields shall be separated by a single occurrence of the field delimiter character. Lines with no field delimiters shall be passed through intact, unless -s is specified. It shall not be an error to select fields not present in the input line.

3858 -n Do not split characters. When specified with the -b option, each
 3859 element in *list* of the form *low-high* (hyphen-separated numbers)
 3860 shall be modified as follows:

3861 If the byte selected by *low* is not the first byte of a charac-
 3862 ter, *low* shall be decremented to select the first byte of the
 3863 character originally selected by *low*. If the byte selected by
 3864 *high* is not the last byte of a character, *high* shall be decre-
 3865 mented to select the last byte of the character prior to the
 3866 character originally selected by *high*, or zero if there is no
 3867 prior character. If the resulting range element has *high*
 3868 equal to zero or *low* greater than *high*, the list element
 3869 shall be dropped from *list* for that input line without caus-
 3870 ing an error.

3871 Each element in list of the form *low-* shall be treated as above
 3872 with *high* set to the the number of bytes in the current line, not
 3873 including the terminating <newline> character. Each element
 3874 in list of the form -*high* shall be treated as above with *low* set to
 3875 1. Each element in list of the form *num* (a single number) shall
 3876 be treated as above with *low* set to *num* and *high* set to *num*.

3877 -s Suppress lines with no delimiter characters, when used with the
 3878 -f option. Unless specified, lines with no delimiters shall be
 3879 passed through untouched.

3880 **4.14.4 Operands**

3881 The following operands shall be supported by the implementation:

3882 *file* A pathname of an input file. If no *file* operands are specified, or if
 3883 a *file* operand is -, the standard input shall be used.

3884 **4.14.5 External Influences**

3885 **4.14.5.1 Standard Input**

3886 The standard input shall be used only if no *file* operands are specified, or if a *file*
 3887 operand is -. See Input Files.

3888 **4.14.5.2 Input Files**

3889 The input files shall be text files, except that line lengths shall be unlimited.

4.14.5.3 Environment Variables

The following environment variables shall affect the execution of `cut`:

LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).
LC_MESSAGES	This variable shall determine the language in which messages should be written.

4.14.5.4 Asynchronous Events

Default.

4.14.6 External Effects

4.14.6.1 Standard Output

The `cut` utility output shall be a concatenation of the selected bytes, characters, or fields (one of the following):

- "%s\n", *<concatenation of bytes>*
- "%s\n", *<concatenation of characters>*
- "%s\n", *<concatenation of fields and field delimiters>*

4.14.6.2 Standard Error

Used only for diagnostic messages.

4.14.6.3 Output Files

None.

4.14.7 Extended Description

None.

4.14.8 Exit Status

The `cut` utility shall exit with one of the following values:

- 0 All input files were output successfully.
- >0 An error occurred.

4.14.9 Consequences of Errors

Default.

4.14.10 Rationale. *(This subclause is not a part of P1003.2)***Examples, Usage**

Examples of the option qualifier list:

- 1, 4, 7 Select the first, fourth, and seventh bytes, characters, or fields and field delimiters.
- 1-3, 8 Equivalent to 1, 2, 3, 8.
- 5, 10 Equivalent to 1, 2, 3, 4, 5, 10.
- 3- Equivalent to third through last.

The *low-high* forms are not always equivalent when used with `-b` and `-n` and multibyte characters. See the description of `-n`.

The following command:

```
cut -d : -f 1,6 /etc/passwd
```

reads the System V password file (user database) and produces lines of the form:

```
<user ID>:<home directory>
```

Most utilities in this standard work on text files. The `cut` utility can be used to turn files with arbitrary line lengths into a set of text files containing the same data. The `paste` utility can be used to create (or recreate) files with arbitrary line lengths. For example, if `file` contains long lines:

```
cut -b 1-500 -n file > file1
cut -b 501- -n file > file2
```

creates `file1` (a text file) with lines no longer than 500 bytes (plus the `<new-line>` character and `file2` that contains the remainder of the data from `file`. (Note that `file2` will not be a text file if there are lines in `file` that are longer than 500 + {`LINE_MAX`} bytes.) The original file can be recreated from `file1` and

3951 file2 using the command:

3952 paste -d "\0" file1 file2 > file

3953 **History of Decisions Made**

3954 Some historical implementations do not count <backspace> characters in deter-
 3955 mining character counts with the `-c` option. This may be useful for using `cut` for
 3956 processing `nroff` output. It was deliberately decided not to have the `-c` option
 3957 treat either <backspace> or <tab> characters in any special fashion. The `fold` 1
 3958 utility does treat these characters specially. 1

3959 Unlike other utilities, some historical implementations of `cut` exit after not
 3960 finding an input file, rather than continuing to process the remaining *file*
 3961 operands. This behavior is prohibited by this standard, where only the exit status
 3962 is affected by this problem.

3963 The behavior of `cut` when provided with either mutually exclusive options or
 3964 options that do not make sense together has been deliberately left unspecified in
 3965 favor of global wording in Section 2.

3966 The traditional `cut` utility has worked in an environment where bytes and char-
 3967 acters were equivalent (modulo <backspace> and <tab> processing in some
 3968 implementations). In the extended world of multibyte characters, the new `-b`
 3969 option has been added. The `-n` option (used with `-b`) allows it to be used to act on
 3970 bytes rounded to character boundaries. The algorithm specified for `-n` guarantees
 3971 that

3972 cut -b 1-500 -n file > file1

3973 cut -b 501- -n file > file2

3974 will end up with all the characters in *file* appearing exactly once in *file1* or
 3975 *file2*. (There is, however, a <newline> character in both *file1* and *file2* for
 3976 each <newline> character in *file*.)

3977 **4.15 date — Write the date and time**

3978 **4.15.1 Synopsis**

3979 `date [-u] [+format]`

3980 **4.15.2 Description**

3981 The `date` utility shall write the date and time to standard output. By default, the
 3982 current date and time shall be written. If an operand beginning with `+` is
 3983 specified, the output format of `date` shall be controlled by the field descriptors
 3984 and other text in the operand.

3985 **4.15.3 Options**

3986 The `date` utility shall conform to the utility argument syntax guidelines
 3987 described in 2.10.2.

3988 The following option shall be supported by the implementation:

3989	<code>-u</code>	Perform operations as if the TZ environment variable was set to	
3990		the string <code>UTC0</code> , or its equivalent historical value of <code>GMT0</code> . Other-	2
3991		wise, <code>date</code> shall use the time zone indicated by the TZ environ-	
3992		ment variable or the system default if that variable is not set.	

3993 **4.15.4 Operands**

3994 When the format is specified, each field descriptor shall be replaced in the stan-
 3995 dard output by its corresponding value. All other characters shall be copied to the
 3996 output without change. The output shall be always terminated with a `<new-`
 3997 `line>` character.

3998 **Field Descriptors**

3999	<code>%a</code>	Locale's abbreviated weekday name.
4000	<code>%A</code>	Locale's full weekday name.
4001	<code>%b</code>	Locale's abbreviated month name.
4002	<code>%B</code>	Locale's full month name.
4003	<code>%c</code>	Locale's appropriate date and time representation.
4004	<code>%C</code>	Century (a year divided by 100 and truncated to an integer) as a
4005		decimal number (00-99).
4006	<code>%d</code>	Day of the month as a decimal number (01-31).
4007	<code>%D</code>	Date in the format <code>mm/dd/yy</code> .
4008	<code>%e</code>	Day of the month as a decimal number (1-31 in a two-digit field
4009		with leading <code><space></code> fill).
4010	<code>%h</code>	A synonym for <code>%b</code> .

4011	%H	Hour (24-hour clock) as a decimal number (00-23).
4012	%I	Hour (12-hour clock) as a decimal number (01-12).
4013	%j	Day of the year as a decimal number (001-366).
4014	%m	Month as a decimal number (01-12).
4015	%M	Minute as a decimal number (00-59).
4016	%n	A <newline> character.
4017	%p	Locale's equivalent of either AM or PM.
4018	%r	12-Hour clock time (01-12) using the <i>AM/PM</i> notation; in the POSIX Locale, this shall be equivalent to "%I:%M:%S %p".
4019		
4020	%S	Seconds as a decimal number (00-61).
4021	%t	A <tab> character.
4022	%T	24-Hour clock time (00-23) in the format <i>HH:MM:SS</i> .
4023	%U	Week number of the year (Sunday as the first day of the week) as a decimal number (00-53).
4024		
4025	%w	Weekday as a decimal number [0 (Sunday)-6].
4026	%W	Week number of the year (Monday as the first day of the week) as a decimal number (00-53).
4027		
4028	%x	Locale's appropriate date representation.
4029	%X	Locale's appropriate time representation.
4030	%Y	Year (offset from %C) as a decimal number (00-99).
4031	%Y	Year with century as a decimal number.
4032	%Z	Time-zone name, or no characters if no time zone is determinable.
4033	%%	A <percent-sign> character.

4034 See the LC_TIME description in 2.5.2.5 for the field descriptor values in the POSIX
 4035 Locale.

4036 **Modified Field Descriptors**

4037 Some field descriptors can be modified by the E and O modifier characters to indi-
 4038 cate a different format or specification as specified in the LC_TIME locale descrip-
 4039 tion (see 2.5.2.5). If the corresponding keyword (see era, era_year, era_d_fmt,
 4040 and alt_digits in 2.5.2.5) is not specified or not supported for the current
 4041 locale, the unmodified field descriptor value shall be used.

4042	%EC	Locale's alternate appropriate date and time representation.
4043	%EC	The name of the base year (period) in the locale's alternate
4044		representation.
4045	%Ex	Locale's alternate date representation.
4046	%Ey	Offset from %EC (year only) in the locale's alternate representa-
4047		tion.
4048	%EY	Full alternate year representation.
4049	%Od	Day of month using the locale's alternate numeric symbols.
4050	%Oe	Day of month using the locale's alternate numeric symbols.
4051	%OH	Hour (24-hour clock) using the locale's alternate numeric symbols.

4052	%OI	Hour (12-hour clock) using the locale's alternate numeric symbols.
4053	%Om	Month using the locale's alternate numeric symbols.
4054	%OM	Minutes using the locale's alternate numeric symbols.
4055	%OS	Seconds using the locale's alternate numeric symbols.
4056	%OU	Week number of the year (Sunday as the first day of the week)
4057		using the locale's alternate numeric symbols.
4058	%Ow	Weekday as number in the locale's alternate representation (Sun-
4059		day = 0).
4060	%OW	Week number of the year (Monday as the first day of the week)
4061		using the locale's alternate numeric symbols.
4062	%Oy	Year (offset from %C) in alternate representation.

4063 **4.15.5 External Influences**

4064 **4.15.5.1 Standard Input**

4065 None.

4066 **4.15.5.2 Input Files**

4067 None.

4068 **4.15.5.3 Environment Variables**

4069 The following environment variables shall affect the execution of `date`:

4070	LANG	This variable shall determine the locale to use for the
4071		locale categories when both LC_ALL and the correspond-
4072		ing environment variable (beginning with LC_) do not
4073		specify a locale. See 2.6.
4074	LC_ALL	This variable shall determine the locale to be used to over-
4075		ride any values for locale categories specified by the set-
4076		tings of LANG or any environment variables beginning
4077		with LC_ .
4078	LC_CTYPE	This variable shall determine the locale for the interpreta-
4079		tion of sequences of bytes of text data as characters (e.g.,
4080		single- versus multibyte characters in arguments).
4081	LC_MESSAGES	This variable shall determine the language in which mes-
4082		sages should be written.

4083 **LC_TIME** This variable shall determine the format and contents of
 4084 date and time strings written by `date`.

4085 **TZ** This variable shall specify the time zone in which the time
 4086 and date are written, unless the `-u` option is specified. If
 4087 the **TZ** variable is not set and the `-u` is not specified, an
 4088 unspecified system default time zone is used.

4089 **4.15.5.4 Asynchronous Events**

4090 Default.

4091 **4.15.6 External Effects**

4092 **4.15.6.1 Standard Output**

4093 When no formatting operand is specified, the output in the POSIX Locale shall be
 4094 equivalent to specifying

4095 `date "+%a %b %e %H:%M:%S %Z %Y"`

4096 **4.15.6.2 Standard Error**

4097 Used only for diagnostic messages.

4098 **4.15.6.3 Output Files**

4099 None.

4100 **4.15.7 Extended Description**

4101 None.

4102 **4.15.8 Exit Status**

4103 The `date` utility shall exit with one of the following values:

4104 0 The date was written successfully.

4105 >0 An error occurred.

4.15.9 Consequences of Errors

Default.

4.15.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The option for setting the date and time was not included. It is normally a system administration option, which is outside the scope of POSIX.2.

The following are input/output examples of `date` used at arbitrary times in the POSIX Locale:

```
$ date
Tue Jun 26 09:58:10 PDT 1990

$ date "+DATE: %m/%d/%y%TIME: %H:%M:%S"
DATE: 11/21/87
TIME: 13:36:16

$ date "+TIME: %r"
TIME: 01:36:32 PM
```

Field descriptors are of unspecified format when not in the POSIX Locale. Some of them can contain <newline>s in some locales, so it may be difficult to use the format shown in Standard Output for parsing the output of `date` in those locales.

The range of values for `%S` extends from 0 to 61 seconds to accommodate the occasional leap second or double leap second.

Although certain of the field descriptors in the POSIX Locale (such as the name of the month) are shown with initial capital letters, this need not be the case in other locales. Programs using these fields may need to adjust the capitalization if the output is going to be used at the beginning of a sentence.

The date string formatting capabilities are intended for use in Gregorian style calendars, possibly with a different starting year (or years). The `%x` and `%c` field descriptors, however, are intended for “local representation”; these may be based on a different, non-Gregorian calendar.

The `%C` field descriptor was introduced to allow a fallback for the `%EC` (alternate year format base year); it can be viewed as the base of the current subdivision in the Gregorian calendar. A century is not calculated as an ordinal number; this standard was approved in century 19, not the twentieth (let’s hope). Both the `%Ey` and `%y` can then be viewed as the offset from `%EC` and `%C`, respectively.

The `E` and `O` modifiers modify the traditional field descriptors, so that they can always be used, even if the implementation (or the current locale) does not support the modifier.

The `E` modifier supports alternate date formats, such as the Japanese Emperor’s Era, as long as these are based on the Gregorian calendar system. Extending the `E` modifiers to other date elements may provide an implementation-specific

extension capable of supporting other calendar systems, especially in combination with the `O` modifier.

The `O` modifier supports time and date formats using the locale's alternate numerical symbols, such as Kanji or Hindi digits, or ordinal number representation.

Non-European locales, whether they use Latin digits in computational items or not, often have local forms of the digits for use in date formats. This is not totally unknown even in Europe; a variant of dates uses Roman numerals for the months: the third day of September 1991 would be written as 3.IX.1991. In Japan, Kanji digits are regularly used for dates; in Arabic-speaking countries, Hindi digits are used. The `%d`, `%e`, `%H`, `%I`, `%m`, `%S`, `%U`, `%w`, `%W`, and `%y` field descriptors always return the date/time field in Latin digits (i.e., 0 through 9). The `%O` modifier was introduced to support the use for display purposes of non-Latin digits. In the `LC_TIME` category in `localedef`, the optional `alt_digits` keyword is intended for this purpose. As an example, assume the following (partial) `localedef` source:

```
alt_digits  "";"I";"II";"III";"IV";"V";"VI";"VII";"VIII" \
            "IX";"X";"XI";"XII"
d_fmt      "%e.%Om.%Y"
```

With the above date, the command

```
date "+x"
```

would yield “3.IX.1991.” With the same `d_fmt`, but without the `alt_digits`, the command would yield “3.9.1991.”

History of Decisions Made

Some of the new options for formatting are from the C Standard {7}. The `-u` option was introduced to allow portable access to Coordinated Universal Time (UTC). The string `GMT0` is allowed as an equivalent `TZ` value to be compatible with all of the systems using the BSD implementation, where this option originated.

The `%e` format field descriptor (adopted from System V) was added because the C Standard {7} descriptors did not provide any way to produce the historical default date output during the first nine days of any month.

4.16 dd — Convert and copy a file

4.16.1 Synopsis

dd [*operand* ...]

4.16.2 Description

The dd utility shall copy the specified input file to the specified output file with possible conversions using specific input and output block sizes. It shall read the input one block at a time, using the specified input block size; it then shall process the block of data actually returned, which could be smaller than the requested block size. It shall apply any conversions that have been specified and write the resulting data to the output in blocks of the specified output block size. If the *bs=expr* operand is specified and no conversions other than *sync* or *noerror* are requested, the data returned from each input block shall be written as a separate output block; if the read returns less than a full block and the *sync* conversion is not specified, the resulting output block shall be the same size as the input block. If the *bs=expr* operand is not specified, or a conversion other than *sync* or *noerror* is requested, the input shall be processed and collected into full-sized output blocks until the end of the input is reached.

The processing order shall be as follows:

- (1) An input block is read.
- (2) If the input block is shorter than the specified input block size and the *sync* conversion is specified, null bytes shall be appended to the input data up to the specified size. The remaining conversions and output shall include the pad characters as if they had been read from the input.
- (3) If the *bs=expr* operand is specified and no conversion other than *sync* or *noerror* is requested, the resulting data shall be written to the output as a single block, and the remaining steps are omitted.
- (4) If the *swab* conversion is specified, each pair of input data bytes shall be swapped. If there are an odd number of bytes in the input block, the results are unspecified.
- (5) Any remaining conversions (*block*, *unblock*, *lcase*, and *ucase*) shall be performed. These conversions shall operate on the input data independently of the input blocking; an input or output fixed-length record may span block boundaries.
- (6) The data resulting from input or conversion or both shall be aggregated into output blocks of the specified size. After the end of input is reached, any remaining output shall be written as a block without padding if *conv=sync* is not specified; thus the final output block may be shorter than the output block size.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

4.16.3 Options

None.

4.16.4 Operands

All of the operands shall be processed before any input is read. The following operands shall be supported by the implementation:

<code>if=file</code>	Specify the input pathname; the default is standard input.
<code>of=file</code>	Specify the output pathname; the default is standard output. If the <code>seek=expr</code> conversion is not also specified, the output file shall be truncated before the copy begins, unless <code>conv=notrunc</code> is specified. If <code>seek=expr</code> is specified, but <code>conv=notrunc</code> is not, the effect of the copy shall be to preserve the blocks in the output file over which <code>dd</code> seeks, but no other portion of the output file shall be preserved. (If the size of the seek plus the size of the input file is less than the previous size of the output file, the output file shall be shortened by the copy.)
<code>ibs=expr</code>	Specify the input block size, in bytes, by <i>expr</i> (default is 512).
<code>obs=expr</code>	Specify the output block size, in bytes, by <i>expr</i> (default is 512).
<code>bs=expr</code>	Set both input and output block sizes to <i>expr</i> bytes, superseding <code>ibs=</code> and <code>obs=</code> . If no conversion other than <code>sync</code> , <code>noerror</code> , and <code>notrunc</code> is specified, each input block shall be copied to the output as a single block without aggregating short blocks.
<code>cbs=expr</code>	Specify the conversion block size for <code>block</code> and <code>unblock</code> in bytes by <i>expr</i> (default is zero). If <code>cbs=</code> is omitted or given a value of zero, using <code>block</code> or <code>unblock</code> produces unspecified results.
<code>skip=n</code>	Skip <i>n</i> input blocks (using the specified input block size) before starting to copy. On seekable files, the implementation shall read the blocks or seek past them; on nonseekable files, the blocks shall be read and the data shall be discarded.
<code>seek=n</code>	Skip <i>n</i> blocks (using the specified output block size) from beginning of output file before copying. On nonseekable files, existing blocks shall be read and space from the current end of file to the specified offset, if any, filled with null bytes; on seekable files, the implementation shall seek to the specified offset or read the blocks as described for nonseekable files.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

4255	<code>count=<i>n</i></code>	Copy only <i>n</i> input blocks.	
4256	<code>conv=<i>value</i>[, <i>value</i> ...]</code>		
4257		Where <i>values</i> are comma-separated symbols from the follow-	
4258		ing list.	
4259	<code>block</code>	Treat the input as a sequence of <newline>-terminated or	2
4260		end-of-file-terminated variable length records independent	2
4261		of the input block boundaries. Each record shall be con-	
4262		verted to a record with a fixed length specified by the	
4263		conversion block size. Any <newline> shall be removed	2
4264		from the input line; <space>s shall be appended to lines	
4265		that are shorter than their conversion block size to fill the	
4266		block. Lines that are longer than the conversion block size	
4267		shall be truncated to the largest number of characters that	
4268		will fit into that size; the number of truncated lines shall be	
4269		reported (see Standard Error below).	
4270		The <code>block</code> and <code>unblock</code> values are mutually exclusive.	
4271	<code>unblock</code>	Convert fixed length records to variable length. Read a	
4272		number of bytes equal to the conversion block size, delete all	
4273		trailing <space>s, and append a <newline>.	2
4274	<code>lcase</code>	Map uppercase characters specified by the LC_CTYPE key-	
4275		word <code>tolower</code> to the corresponding lowercase character.	
4276		Characters for which no mapping is specified shall not be	
4277		modified by this conversion.	
4278		The <code>lcase</code> and <code>ucase</code> symbols are mutually exclusive.	
4279	<code>ucase</code>	Map lowercase characters specified by the LC_CTYPE key-	
4280		word <code>toupper</code> to the corresponding uppercase character.	
4281		Characters for which no mapping is specified shall not be	
4282		modified by this conversion.	
4283	<code>swab</code>	Swap every pair of input bytes.	
4284	<code>noerror</code>	Do not stop processing on an input error. When an input	
4285		error occurs, a diagnostic message shall be written on stan-	
4286		dard error, followed by the current input and output block	
4287		counts in the same format as used at completion (see Stan-	
4288		dard Error). If the <code>sync</code> conversion is specified, the missing	
4289		input shall be replaced with null bytes and processed nor-	
4290		mally; otherwise, the input block shall be omitted from the	
4291		output.	
4292	<code>notrunc</code>	Do not truncate the output file. Preserve blocks in the out-	
4293		put file not explicitly written by this invocation of the <code>dd</code>	
4294		utility. (See also the preceding <code>of=file</code> operand.)	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

4295 `sync` Pad every input block to the size of `ibs=` buffer, appending
 4296 null bytes.

4297 The behavior is unspecified if operands other than `conv=` are specified more than
 4298 once.

4299 For the `bs=`, `cbs=`, `ibs=`, and `obs=` operands, the application shall supply an
 4300 expression specifying a size in bytes. The expression, *expr*, can be:

- 4301 (1) a positive decimal number;
- 4302 (2) a positive decimal number followed by `k`, specifying multiplication by
 4303 1024;
- 4304 (3) a positive decimal number followed by `b`, specifying multiplication by
 4305 512; or
- 4306 (4) two or more positive decimal numbers (with or without `k` or `b`) separated
 4307 by `x`, specifying the product of the indicated values.

4308 **4.16.5 External Influences**

4309 **4.16.5.1 Standard Input**

4310 If no `if=` operand is specified, the standard input shall be used. See Input Files.

4311 **4.16.5.2 Input Files**

4312 The input file can be any file type.

4313 **4.16.5.3 Environment Variables**

4314 The following environment variables shall affect the execution of `dd`:

4315 **LANG** This variable shall determine the locale to use for the
 4316 locale categories when both **LC_ALL** and the correspond-
 4317 ing environment variable (beginning with **LC_**) do not
 4318 specify a locale. See 2.6.

4319 **LC_ALL** This variable shall determine the locale to be used to over-
 4320 ride any values for locale categories specified by the set-
 4321 tings of **LANG** or any environment variables beginning
 4322 with **LC_**.

4323 **LC_CTYPE** This variable shall determine the locale for the interpreta-
 4324 tion of sequences of bytes of text data as characters (e.g.,
 4325 single- versus multibyte characters in arguments and
 4326 input files), the classification of characters as upper- or
 4327 lowercase, and the mapping of characters from one case to
 4328 the other.

4329 **LC_MESSAGES** This variable shall determine the language in which mes-
 4330 sages should be written.

4331 **4.16.5.4 Asynchronous Events**

4332 For SIGINT, the `dd` utility shall write status information to standard error before
 4333 exiting. It shall take the standard action for all other signals; see 2.11.5.4.

4334 **4.16.6 External Effects**

4335 **4.16.6.1 Standard Output**

4336 If no `of=` operand is specified, the standard output shall be used. The nature of
 4337 the output depends on the operands selected.

4338 **4.16.6.2 Standard Error**

4339 On completion, `dd` shall write the number of input and output blocks to standard
 4340 error. In the POSIX Locale the following formats shall be used:

4341 "%u+%u records in\n", *<number of whole input blocks>*,
 4342 *<number of partial input blocks>*

4343 "%u+%u records out\n", *<number of whole output blocks>*,
 4344 *<number of partial output blocks>*

4345 A partial input block is one for which `read()` returned less than the input block
 4346 size. A partial output block is one that was written with fewer bytes than
 4347 specified by the output block size.

4348 In addition, when there is at least one truncated block, the number of truncated
 4349 blocks shall be written to standard error. In the POSIX Locale, the format shall
 4350 be:

4351 "%u truncated %s\n", *<number of truncated blocks>*, "block" [if
 4352 *<number of truncated blocks>* is one] "blocks" [otherwise]

4353 Diagnostic messages may also be written to standard error.

4354 **4.16.6.3 Output Files**

4355 If the `of=` operand is used, the output shall be the same as described in Standard
 4356 Output.

4.16.7 Extended Description

None.

4.16.8 Exit Status

The `dd` utility shall exit with one of the following values:

0 The input file was copied successfully.

>0 An error occurred.

4.16.9 Consequences of Errors

If an input error is detected and the `noerror` conversion has not been specified, any partial output block shall be written to the output file, a diagnostic message shall be written, and the copy operation shall be discontinued. If some other error is detected, a diagnostic message shall be written and the copy operation shall be discontinued.

4.16.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The input and output block size can be specified to take advantage of raw physical I/O.

The following command:

```
dd if=/dev/rmt0h of=/dev/rmt1h
```

copies from tape drive 0 to tape drive 1, using a common historical device naming convention.

The following command:

```
dd ibs=10 skip=1
```

strips the first 10 bytes from standard input.

A suggested implementation technique for `conv=noerror, sync` is to zero the input buffer before each read and to write the contents of the input buffer to the output even after an error. In this manner, any data transferred to the input buffer before the error was detected will be preserved. Another point is that a failed read on a regular file or a disk will generally not increment the file offset, and `dd` must then seek past the block on which the error occurred; otherwise, the input error will occur repetitively. When the input is a magnetic tape, however, the tape will normally have passed the block containing the error when the error is reported, and thus no seek is necessary.

History of Decisions Made**Table 4-4 – ASCII to EBCDIC Conversion**

	<u>0</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>
0000	0000	0001	0002	0003	0067	0055	0056	0057
0010	0026	0005	0045	0013	0014	0015	0016	0017
0020	0020	0021	0022	0023	0074	0075	0062	0046
0030	0030	0031	0077	0047	0034	0035	0036	0037
0040	0100	0132	0177	0173	0133	0154	0120	0175
0050	0115	0135	0134	0116	0153	0140	0113	0141
0060	0360	0361	0362	0363	0364	0365	0366	0367
0070	0370	0371	0172	0136	0114	0176	0156	0157
0100	0174	0301	0302	0303	0304	0305	0306	0307
0110	0310	0311	0321	0322	0323	0324	0325	0326
0120	0327	0330	0331	0342	0343	0344	0345	0346
0130	0347	0350	0351	0255	0340	0275	<u>0232</u>	0155
0140	0171	0201	0202	0203	0204	0205	0206	0207
0150	0210	0211	0221	0222	0223	0224	0225	0226
0160	0227	0230	0231	0242	0243	0244	0245	0246
0170	0247	0250	0251	0300	0117	0320	<u>0137</u>	0007
0200	0040	0041	0042	0043	0044	0025	0006	0027
0210	0050	0051	0052	0053	0054	0011	0012	0033
0220	0060	0061	0032	0063	0064	0065	0066	0010
0230	0070	0071	0072	0073	0004	0024	0076	0341
0240	0101	0102	0103	0104	0105	0106	0107	0110
0250	0111	0121	0122	0123	0124	0125	0126	0127
0260	0130	0131	0142	0143	0144	0145	0146	0147
0270	0150	0151	0160	0161	0162	0163	0164	0165
0300	0166	0167	0170	0200	0212	0213	0214	0215
0310	0216	0217	0220	<u>0152</u>	0233	0234	0235	0236
0320	0237	0240	0252	0253	0254	<u>0112</u>	0256	0257
0330	0260	0261	0262	0263	0264	0265	0266	0267
0340	0270	0271	0272	0273	0274	<u>0241</u>	0276	0277
0350	0312	0313	0314	0315	0316	0317	0332	0333
0360	0334	0335	0336	0337	0352	0353	0354	0355
0370	0356	0357	0372	0373	0374	0375	0376	0377

The Options subclause is listed as “None” because there are no options recognized by historical `dd` utilities. Certainly, many of the operands could have been designed to use the Utility Syntax Guidelines, which would have resulted in the classic hyphenated option letters. In this version of this standard, `dd` retains its curious JCL-like syntax due to the large number of applications that depend on the historical implementation. “Fixing” the interface would cause an excessive compatibility problem. However, due to interest in the international community, the developers of the standard have agreed to provide an alternative syntax for

Table 4-5 – ASCII to IBM EBCDIC Conversion

		<u>0</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>
	0000	0000	0001	0002	0003	0067	0055	0056	0057
	0010	0026	0005	0045	0013	0014	0015	0016	0017
	0020	0020	0021	0022	0023	0074	0075	0062	0046
	0030	0030	0031	0077	0047	0034	0035	0036	0037
	0040	0100	0132	0177	0173	0133	0154	0120	0175
	0050	0115	0135	0134	0116	0153	0140	0113	0141
	0060	0360	0361	0362	0363	0364	0365	0366	0367
	0070	0370	0371	0172	0136	0114	0176	0156	0157
	0100	0174	0301	0302	0303	0304	0305	0306	0307
	0110	0310	0311	0321	0322	0323	0324	0325	0326
	0120	0327	0330	0331	0342	0343	0344	0345	0346
	0130	0347	0350	0351	0255	0340	0275	<u>0137</u>	0155
	0140	0171	0201	0202	0203	0204	0205	0206	0207
	0150	0210	0211	0221	0222	0223	0224	0225	0226
	0160	0227	0230	0231	0242	0243	0244	0245	0246
	0170	0247	0250	0251	0300	0117	0320	<u>0241</u>	0007
	0200	0040	0041	0042	0043	0044	0025	0006	0027
	0210	0050	0051	0052	0053	0054	0011	0012	0033
	0220	0060	0061	0032	0063	0064	0065	0066	0010
	0230	0070	0071	0072	0073	0004	0024	0076	0341
	0240	0101	0102	0103	0104	0105	0106	0107	0110
	0250	0111	0121	0122	0123	0124	0125	0126	0127
	0260	0130	0131	0142	0143	0144	0145	0146	0147
	0270	0150	0151	0160	0161	0162	0163	0164	0165
	0300	0166	0167	0170	0200	0212	0213	0214	0215
	0310	0216	0217	0220	<u>0232</u>	0233	0234	0235	0236
	0320	0237	0240	0252	0253	0254	<u>0255</u>	0256	0257
	0330	0260	0261	0262	0263	0264	0265	0266	0267
	0340	0270	0271	0272	0273	0274	<u>0275</u>	0276	0277
	0350	0312	0313	0314	0315	0316	0317	0332	0333
	0360	0334	0335	0336	0337	0352	0353	0354	0355
	0370	0356	0357	0372	0373	0374	0375	0376	0377

the next version of this standard that conforms to the spirit of the Utility Syntax Guidelines. This new syntax will be accompanied by the existing syntax, marked as obsolescent. System implementors are encouraged to develop and promulgate a new syntax for `dd`, perhaps using a different utility name, that can be adopted for the next version of this standard.

The default `ibs=` and `obs=` sizes are specified as 512 bytes because there are existing (largely portable) scripts that assume these values. If they were left unspecified, very strange results could occur if an implementation chose an odd block size.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

Historical implementations of `dd` used `creat()` when processing `of=file`. This makes the `seek=` operand unusable except on special files. More recent BSD-based implementations use `open()` (without `O_TRUNC`) instead of `creat()`, but fail to delete output file contents after the data copied. Since balloting showed a desire to make this behavior available, the `conv=notrunc` feature was added.

The `w` multiplier, (historically meaning *word*), is used in System V to mean 2 and in 4.2BSD to mean 4. Since *word* is inherently nonportable, its use is not supported by POSIX.2.

All references to US ASCII and to conversions to/from IBM and EBCDIC were removed in preparation for this document's acceptance by the international community. Implementations are free to have such conversions as extensions, using the `ascii`, `ibm`, and `ebcdic` keywords. However, in the interest of promoting consistency of implementation, the original material from an early draft has been restored to the rationale as an example:

In the two tables, the conversions from ASCII to either standard EBCDIC (Table 4-4) or the IBM version of EBCDIC (Table 4-5) are shown. The differences between the two tables are underlined. In both tables, the ASCII values are the row and column headers and the EBCDIC values are found at their intersections. For example, ASCII 0012 (LF) is the second row, third column, yielding 0045 in EBCDIC. The inverted tables (for EBCDIC to ASCII conversion) are not shown, but are in one-to-one correspondence with these tables. The tables are understood to match recent System V conversion algorithms and there have been reports that earlier System V versions and the BSD version do not always conform to these; however, representatives of the BSD development group have agreed that a future version of their system will use these tables for consistency with System V.

The `cbs` operand is required if any of the `ascii`, `ebcdic`, or `ibm` operands are specified. For the `ascii` operand, the input is handled as described for the `unblock` operand except that characters are converted to ASCII before the trailing `<spaces>`s are deleted. For the `ebcdic` and `ibm` operands, the input is handled as described for the `block` operand except that the characters are converted to EBCDIC or IBM EBCDIC after the trailing `<spaces>`s are added.

The `block` and `unblock` keywords are from historical BSD practice.

Early drafts only allowed two numbers separated by `x` to be used in a product when specifying `bs=`, `cbs=`, `ibs=`, and `obs=` sizes. This was changed to reflect the historical practice of allowing multiple numbers in the product as provided by Version 7 and all releases of System V and BSD.

4.17 diff — Compare two files

4.17.1 Synopsis

```
diff [-c | -e | -C n] [-br] file1 file2
```

4.17.2 Description

The `diff` utility shall compare the contents of *file1* and *file2* and write to standard output a list of changes necessary to convert *file1* into *file2*. This list should be minimal. No output shall be produced if the files are identical.

4.17.3 Options

The `diff` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

- `-b` Cause trailing <blank>s to be ignored and other strings of <blank>s to compare equal.
- `-c` Produce output in a form that provides three lines of context.
- `-C n` Produce output in a form that provides *n* lines of context (where *n* shall be interpreted as a positive decimal integer).
- `-e` Produce output in a form suitable as input for the `ed` utility (see 4.20), which can then be used to convert *file1* into *file2*.
- `-r` Apply `diff` recursively to files and directories of the same name when *file1* and *file2* are both directories.

4.17.4 Operands

The following operands shall be supported by the implementation:

- file1*
file2 A pathname of a file to be compared. If either the *file1* or *file2* operand is `-`, the standard input shall be used in its place.

If both *file1* and *file2* are directories, `diff` shall not compare block special files, character special files, or FIFO special files to any files and shall not compare regular files to directories. The system documentation shall specify the behavior of `diff` on implementation-specific file types not specified by POSIX.1 {8} when found in directories. Further details are as specified in 4.17.6.1.1.

If only one of *file1* and *file2* is a directory, `diff` shall be applied to the nondirectory file and the file contained in the directory file with a filename that is the same as the last component of the nondirectory file.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

4.17.5 External Influences

4.17.5.1 Standard Input

The standard input shall be used only if one of the *file1* or *file2* operands references standard input. See Input Files.

4.17.5.2 Input Files

The input files shall be text files.

4.17.5.3 Environment Variables

The following environment variables shall affect the execution of `diff`:

LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).
LC_MESSAGES	This variable shall determine the language in which messages should be written.
LC_TIME	This variable shall determine the locale for affecting the format of file time stamps written with the <code>-C</code> and <code>-c</code> options.
TZ	This variable shall determine the locale for affecting the time zone used for calculating file time stamps written with the <code>-C</code> and <code>-c</code> options.

4.17.5.4 Asynchronous Events

Default.

4.17.6 External Effects

4.17.6.1 Standard Output

4.17.6.1.1 `diff` Directory Comparison Format

If both *file1* and *file2* are directories, the following output formats shall be used.

In the POSIX Locale, each file that is present in only one directory shall be reported using the following format:

"Only in %s: %s\n", <directory pathname>, <filename>

In the POSIX Locale, subdirectories that are common to the two directories may be reported with the following format:

"Common subdirectories: %s and %s\n", <directory1 pathname>,
<directory2 pathname>

For each file common to the two directories if the two files are not to be compared, the following format shall be used in the POSIX Locale:

"File %s is a %s while file %s is a %s\n",
<directory1 pathname>, <file type of directory1 pathname>,
<directory2 pathname>, <file type of directory2 pathname>

For each file common to the two directories, if the files are to be compared and are identical, no output shall be written. If the two files differ, the following format shall be written: 2

"diff %s %s %s\n", <diff_options>, <filename1>, <filename2>

where <diff_options> are the options as specified on the command line. Depending on these options, one of the following output formats shall be used to write the differences.

All directory pathnames listed in this subclause shall be relative to the original command line arguments. All other names of files listed in this subclause shall be filenames (pathname components).

4.17.6.1.2 `diff` Default Output Format

The default (without `-e`, `-c`, or `-C` options) `diff` utility output contains lines of these forms:

"%da%d\n", <num1>, <num2>

"%da%d,%d\n", <num1>, <num2>, <num3>

"%dd%d\n", <num1>, <num2>

"%d,%dd%d\n", <num1>, <num2>, <num3>

"%dc%d\n", <num1>, <num2>

"%d,%dc%d\n", <num1>, <num2>, <num3>

4615 "%dc%d, %d\n", <num1>, <num2>, <num3>

4616 "%d, %dc%d, %d\n", <num1>, <num2>, <num3>, <num4>

4617 These lines resemble `ed` subcommands to convert *file1* into *file2*. The line
 4618 numbers before the action letters shall pertain to *file1*; those after shall pertain to
 4619 *file2*. Thus, by exchanging 'a' for 'd' and reading the line in reverse order, one
 4620 can also determine how to convert *file2* into *file1*. As in `ed`, identical pairs (where
 4621 *num1* = *num2*) are abbreviated as a single number.

4622 Following each of these lines, `diff` shall write to standard output all lines
 4623 affected in the first file using the format:

4624 "<Δ%S ", <line>

4625 and all lines affected in the second file using the format:

4626 ">Δ%S ", <line>

4627 If there are lines affected in both *file1* and *file2* (as with the `c` subcommand), the
 4628 changes are separated with a line consisting of three hyphens:

4629 " ---\n"

4630 4.17.6.1.3 `diff -e` Output Format

4631 With the `-e` option, a script shall be produced that shall, when provided as input
 4632 to `ed` (see 4.20), along with an appended `w` (write) command, convert *file1* into
 4633 *file2*. Only the `a` (append), `c` (change), `d` (delete), `i` (insert), and `s` (substitute) com-
 4634 mands of `ed` shall be used in this script. Text line(s), except those consisting of
 4635 the single character period (.), shall be output as they appear in the file.

4636 4.17.6.1.4 `diff -c` or `-C` Output Format

4637 With the `-c` or `-C` option, the output format shall consist of affected lines along
 4638 with surrounding lines of context. The affected lines shall show which ones need
 4639 to be deleted or changed in *file1*, and those added from *file2*. With the `-c` option,
 4640 three lines of context, if available, shall be written before and after the affected
 4641 lines. With the `-C` option, the user can specify how many lines of context shall be
 4642 written. The exact format follows.

4643 The name and last modification time of each file shall be output in the following
 4644 format:

4645 "*** %S %S\n", *file1*, <file1 time stamp>

4646 " --- %S %S\n", *file2*, <file2 time stamp>

4647 and a string of 15 asterisks:

4648 " *****\n"

4649 Each <file> field shall be the pathname of the corresponding file being compared.
 4650 The pathname written for standard input is unspecified.

4651 In the POSIX Locale, each <time stamp> field shall be equivalent to the output
 4652 from the following command:

Copyright © 1991 IEEE. All rights reserved.
 This is an unapproved IEEE Standards Draft, subject to change.

4653 date "+%a %b %e %T %Y"

4654 without the trailing <newline>, executed at the time of last modification of the
4655 corresponding file (or the current time, if the file is standard input).

4656 Then, the following output formats shall be applied for every set of changes.

4657 First, the range of lines in *file1* shall be written in the following format:

4658 "*** %d, %d ****\n", <beginning line number>, <ending line number>

4659 Next, the affected lines along with lines of context (unaffected lines) shall be writ-
4660 ten. Unaffected lines shall be written in the following format:

4661 "ΔΔ%S", <unaffected_line>

4662 Deleted lines shall be written as:

4663 "-Δ%S", <deleted_line>

4664 Changed lines shall be written as:

4665 "!Δ%S", <changed_line>

4666 Next, the range of lines in *file2* shall be written in the following format:

4667 "--- %d, %d ----\n", <beginning line number>, <ending line number>

4668 Then, lines of context and changed lines shall be written as described in the previ-
4669 ous formats. Lines added from *file2* shall be written in the following format:

4670 "+Δ%S", <added_line>

4671 **4.17.6.2 Standard Error**

4672 Used only for diagnostic messages.

4673 **4.17.6.3 Output Files**

4674 None.

4675 **4.17.7 Extended Description**

4676 None.

4677 **4.17.8 Exit Status**

4678 The `diff` utility shall exit with one of the following values:

4679 0 No differences were found.

4680 1 Differences were found.

4681 >1 An error occurred.

4.17.9 Consequences of Errors

Default.

4.17.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

If lines at the end of a file are changed and other lines are added, `diff` output may show this as a delete and add, as a change, or as a change and add; `diff` is not expected to know which happened and users should not care about the difference in output as long as it clearly shows the differences between the files.

If `dir1` is a directory containing a directory named `x`, `dir2` is a directory containing a directory named `x`, `dir1/x` and `dir2/x` both contain files named `date.out`, and `dir2/x` contains a file named `y`, the command:

```
diff -r dir1 dir2
```

could produce output similar to:

```
Common subdirectories: dir1/x and dir2/x
Only in dir2/x: y
diff -r dir1/x/date.out dir2/x/date.out
lcl
< Mon Jul  2 13:12:16 PDT 1990
---
> Tue Jun 19 21:41:39 PDT 1990
```

History of Decisions Made

The `-h` option was removed because it was insufficiently specified and it does not add to application portability.

Current implementations employ algorithms that do not always produce a minimum list of differences; the current language about making every effort is the best the standard can do, as there is no metric that could be employed to judge the quality of implementations against any and all file contents. The statement “This list should be minimal” clearly implies that implementations are not expected to provide the following output when comparing two 100-line files that differ in only one character on a single line:

```
1,100c1,100
all 100 lines from file1 preceded with "< "
---
all 100 lines from file2 preceded with "> "
```

The “Only in” messages required by this standard when the `-r` option is specified, is not used by most historical implementations if the `-e` option is also specified. It is required here because it provides useful information that must be provided to update a target directory hierarchy to match a source hierarchy. The “Common subdirectories” messages are written by System V and 4.3BSD when the `-r` option is specified. They are allowed here, but are not required because they are

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

reporting on something that is the same, not reporting a difference, and are not needed to update a target hierarchy.

The `-c` option, which writes output in a format using lines of context, has been included. The format is useful for a variety of reasons, among them being much improved readability, and the ability to understand difference changes when the target file has line numbers that differ from another similar, but slightly different, copy. An important utility, `patch`, which has proved itself indispensable to the USENET community, often only works with difference listings using the context format. The BSD version of `-c` takes an optional argument specifying the amount of context. Rather than overloading `-c` and breaking the Utility Syntax Guidelines for `diff`, the working group decided to add a separate option for specifying a context diff with a specified amount of context (`-C`). Also, the format for context diffs was extended slightly in 4.3BSD to allow multiple changes that are within context lines from each other to be merged together. The output format contains an additional four asterisks after the range of affected lines in the first filename. This was to provide a flag for old programs (like old versions of `patch`) that only understand the old context format. The version of context described here does not require that multiple changes within context lines be merged, but does not prohibit it either. The extension is upward compatible, so any vendors that wish to retain the old version of `diff` can do so by just adding the extra four asterisks (that is, utilities that currently use `diff` and understand the new merged format will also understand the old unmerged format, but not vice-versa).

The substitute command was added as an additional format for the `-e` option. This was added to provide implementations a way to fix the classic “dot alone on a line” bug present in many versions of `diff`. Since many implementations have fixed this bug the working group decided not to standardize broken behavior, but rather, provide the necessary tool for fixing the bug. One way to fix this bug is to output two periods whenever a lone period is needed, then terminate the append command with a period, and then use the substitute command to convert the two periods into one period.

The `-f` flag was not included as it provides no additional functionality over the `-e` option.

The BSD-derived `-r` option was added to provide a mechanism for using `diff` to compare two file system trees. This behavior is useful, is standard practice on all BSD-derived systems, and is not easily reproducible with the `find` utility.

The requirement that `diff` not compare files in some circumstances, even though they have the same name, was added in response to ballot objections and digging further into the actual output of historical implementations. The message specified here is already in use when a directory is being compared to a nondirectory. It is extended here to preclude the problems arising from running into FIFOs and other files that would cause `diff` to hang waiting for input with no indication to the user that `diff` was hung. In most common usage, `diff -r` should indicate differences in the file hierarchies, not the difference of contents of devices pointed to by the hierarchies.

4766 Many early implementations of `diff` require seekable files. Since POSIX.1 {8}
 4767 supports named pipes, the working group decided that such a restriction was
 4768 unreasonable. Note also that the allowed file name – almost always refers to a
 4769 pipe.

4770 No directory search order is being specified in 4.17.6.1.1. The historical ordering
 4771 is, in fact, not optimal, in that it prints out all of the differences at the current
 4772 level, including the statements about all common subdirectories before recursing
 4773 into those subdirectories.

4774 The message 2

4775 `"diff %s %s %s\n", <diff_options>, <filename1>, <filename2>` 2

4776 does not vary by locale because it is the representation of a command, not an 2
 4777 English sentence. 2

4778 **4.18 `dirname` — Return directory portion of pathname**

4779 **4.18.1 Synopsis**

4780 `dirname string`

4781 **4.18.2 Description**

4782 The *string* operand shall be treated as a pathname, as defined in 2.2.2.102. The
 4783 *string string* shall be converted to the name of the directory containing the
 4784 filename corresponding to the last pathname component in *string*, performing
 4785 actions equivalent to the following steps in order:

- 4786 (1) If *string* is `//`, skip steps (2) through (5).
- 4787 (2) If *string* consists entirely of slash characters, *string* shall be set to a sin-
 4788 gle slash character. In this case, skip steps (3) through (8).
- 4789 (3) If there are any trailing slash characters in *string*, they shall be removed.
- 4790 (4) If there are no slash characters remaining in *string*, *string* shall be set to
 4791 a single period character. In this case, skip steps (5) through (8).
- 4792 (5) If there are any trailing nonslash characters in *string*, they shall be
 4793 removed.
- 4794 (6) If the remaining *string* is `//`, it is implementation defined whether steps
 4795 (7) and (8) are skipped or processed.
- 4796 (7) If there are any trailing slash characters in *string*, they shall be removed.
- 4797 (8) If the remaining *string* is empty, *string* shall be set to a single slash char-
 4798 acter.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

4799 The resulting string shall be written to standard output.

4800 **4.18.3 Options**

4801 None.

4802 **4.18.4 Operands**

4803 The following operand shall be supported by the implementation:

4804 *string* A string.

4805 **4.18.5 External Influences**

4806 **4.18.5.1 Standard Input**

4807 None.

4808 **4.18.5.2 Input Files**

4809 None.

4810 **4.18.5.3 Environment Variables**

4811 The following environment variables shall affect the execution of `dirname`:

4812 **LANG** This variable shall determine the locale to use for the
4813 locale categories when both **LC_ALL** and the correspond-
4814 ing environment variable (beginning with **LC_**) do not
4815 specify a locale. See 2.6.

4816 **LC_ALL** This variable shall determine the locale to be used to over-
4817 ride any values for locale categories specified by the set-
4818 tings of **LANG** or any environment variables beginning
4819 with **LC_**.

4820 **LC_CTYPE** This variable shall determine the locale for the interpreta-
4821 tion of sequences of bytes of text data as characters (e.g.,
4822 single- versus multibyte characters in arguments).

4823 **LC_MESSAGES** This variable shall determine the language in which mes-
4824 sages should be written.

4825 **4.18.5.4 Asynchronous Events**

4826 Default.

4.18.6 External Effects

4.18.6.1 Standard Output

The `dirname` utility shall write a line to the standard output in the following format:

```
"%s\n", <resulting string>
```

4.18.6.2 Standard Error

Used only for diagnostic messages.

4.18.6.3 Output Files

None.

4.18.7 Extended Description

None.

4.18.8 Exit Status

The `dirname` utility shall exit with one of the following values:

0 Successful completion.

>0 An error occurred.

4.18.9 Consequences of Errors

Default.

4.18.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The `dirname` utility originated in System III. It has evolved through the System V releases to a version that matches the requirements specified in this description in System V Release 3.

4.3BSD and earlier versions did not include `dirname`.

Table 4-6 indicates the results required for some invocations of `dirname`.

Table 4-6 – dirname Examples

Command	Results
dirname /	/
dirname //	/ or //
dirname /a/b/	/a
dirname //a//b//	//a
dirname	<i>unspecified</i>
dirname a	. (\$? = 0)
dirname ""	. (\$? = 0)
dirname /a	/
dirname /a/b	/a
dirname a/b	a

History of Decisions Made

The behaviors of `basename` and `dirname` in this standard have been coordinated so that when *string* is a valid pathname

```
$(basename "string")
```

would be a valid filename for the file in the directory

```
$(dirname "string")
```

This would not work for the versions of these utilities in earlier drafts due to the way processing of trailing slashes was specified. Consideration was given to leaving processing unspecified if there were trailing slashes, but this cannot be done; the POSIX.1 {8} definition of pathname allows trailing slashes. The `basename` and `dirname` utilities have to specify consistent handling for all valid pathnames.

Since the definition of *pathname* in 2.2.2.102 specifies implementation-defined behavior for pathnames starting with two slash characters, Draft 11 has been changed to specify similar implementation-defined behavior for the `basename` and `dirname` utilities. On implementations where the pathname `//` is always treated the same as the pathname `/`, the functionality required by Draft 10 meets all of the Draft 11 requirements.

4.19 **echo** — Write arguments to standard output

4.19.1 Synopsis

echo [*string* ...]

4.19.2 Description

The **echo** utility shall write its arguments to standard output, followed by a <newline> character. If there are no arguments, only the <newline> character shall be written.

4.19.3 Options

The **echo** utility shall not recognize the **--** argument in the manner specified by utility syntax guideline 10 in 2.10.2; **--** shall be recognized as a string operand.

Implementations need not support any options.

4.19.4 Operands

The following operands shall be supported by the implementation:

<i>string</i>	A string to be written to standard output. If the first operand is "-n" or if any of the operands contain a backslash (\) character, the results are implementation defined.
---------------	--

4.19.5 External Influences

4.19.5.1 Standard Input

None.

4.19.5.2 Input Files

None.

4.19.5.3 Environment Variables

The following environment variables shall affect the execution of **echo**:

LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
-------------	---

4909 **LC_ALL** This variable shall determine the locale to be used to over-
 4910 ride any values for locale categories specified by the set-
 4911 tings of **LANG** or any environment variables beginning
 4912 with **LC_**.

4913 **LC_MESSAGES** This variable shall determine the language in which diag-
 4914 nostic messages should be written.

4915 **4.19.5.4 Asynchronous Events**

4916 Default.

4917 **4.19.6 External Effects**

4918 **4.19.6.1 Standard Output**

4919 The **echo** utility arguments shall be separated by single <space>s and a <new-
 4920 line> character shall follow the last argument.

4921 **4.19.6.2 Standard Error**

4922 Used only for diagnostic messages.

4923 **4.19.6.3 Output Files**

4924 None.

4925 **4.19.7 Extended Description**

4926 None.

4927 **4.19.8 Exit Status**

4928 The **echo** utility shall exit with one of the following values:

4929 0 Successful completion.

4930 >0 An error occurred.

4931 **4.19.9 Consequences of Errors**

4932 Default.

4.19.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

As specified by this standard, `echo` writes its arguments in the simplest of ways. The two different historical versions of `echo` vary in fatal incompatible ways.

The BSD `echo` checks the first argument for the string `"-n"`, which causes it to suppress the `<newline>` character that would otherwise follow the final argument in the output.

The System V `echo` does not support any options, but allows escape sequences within its operands:

`\a` Write an `<alert>` character.

`\b` Write a `<backspace>` character.

`\c` Suppress the `<newline>` character that otherwise follows the final argument in the output. All characters following the `\c` in the arguments are ignored.

`\f` Write a `<form-feed>` character.

`\n` Write a `<newline>` character.

`\r` Write a `<carriage-return>` character.

`\t` Write a `<tab>` character.

`\v` Write a `<vertical-tab>` character.

`\\` Write a backslash character.

`\0num`

Write an 8-bit value that is the 1-, 2-, or 3-digit octal number *num*.

It is not possible to use `echo` portably across these two implementations unless both `-n` (as the first argument) and escape sequences are omitted.

The `printf` utility (see 4.50) can be used to portably emulate any of the traditional behaviors of the `echo` utility as follows:

— The System V `echo` is equivalent to:

```
printf "%b\n" "$*
```

— The BSD `echo` is equivalent to:

```
if [ "X$1" = "X-n" ]
then
    shift
    printf "%s" "$*"
else
    printf "%s\n" "$*"
fi
```

The `echo` utility does not support utility syntax guideline 10 because existing applications depend on `echo` to echo *all* of its arguments, except for the `-n` option

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

4971 in the BSD version.

4972 New applications are encouraged to use `printf` instead of `echo`. The `echo` util-
 4973 ity has not been made obsolescent because of its extremely widespread use in
 4974 existing applications.

4975 **History of Decisions Made**

4976 In Draft 8, an attempt was made to merge the extensions of BSD and System V,
 4977 supporting both `-n` and escape sequences. During initial ballot resolution, a `-e`
 4978 option was proposed to enable the escape conventions. Both attempts failed, as
 4979 there are historical scripts that would be broken by any attempt at reconciliation.
 4980 Therefore, in Draft 9 only the simplest version of `echo` is presented.
 4981 Implementation-defined extensions on BSD and System V will keep historical
 4982 applications content. Portable applications that wish to do prompting without
 4983 `<newline>`s or that could possibly be expecting to echo a `"-n"`, should use the
 4984 new `printf` utility (see 4.50), derived from the Ninth Edition.

4985 The `LC_CTYPE` variable is not cited because `echo`, as specified here, does not
 4986 need to understand the characters in its arguments. The System V and BSD
 4987 implementations might need to be sensitive to it because of their extensions.

4988 **4.20 ed — Edit text**

4989 **4.20.1 Synopsis**

4990 `ed [-p string] [-s] [file]`

4991 *Obsolescent Version:*

4992 `ed [-p string] [-] [file]`

4993 **4.20.2 Description**

4994 The `ed` utility is a line-oriented text editor that shall use two modes: *command*
 4995 *mode* and *input mode*. In command mode the input characters shall be inter-
 4996 preted as commands, and in input mode they shall be interpreted as text. See
 4997 4.20.7.

4998 **4.20.3 Options**

4999 The `ed` utility shall conform to the utility argument syntax guidelines described
 5000 in 2.10.2, except for its nonstandard usage of `-` in the obsolescent version.

5001 The following options shall be supported by the implementation:

- 5002 **-p *string*** Use *string* as the prompt string when in command mode. By
 5003 default, there shall be no prompt string.
- 5004 **-s** Suppress the writing of byte counts by *e*, *E*, *r*, and *w* commands
 5005 and of the *!* prompt after a *!command*.
- 5006 **-** (Obsolescent.) Equivalent to the **-s** option.

5007 **4.20.4 Operands**

5008 The following operand shall be supported by the implementation:

- 5009 ***file*** If the *file* argument is given, *ed* shall simulate an *e* command on
 5010 the file named by the pathname, *file*, before accepting commands
 5011 from the standard input.

5012 **4.20.5 External Influences**

5013 **4.20.5.1 Standard Input**

5014 The standard input shall be a text file consisting of commands, as described in
 5015 4.20.7.

5016 **4.20.5.2 Input Files**

5017 The input files shall be text files.

5018 **4.20.5.3 Environment Variables**

5019 The following environment variables shall affect the execution of *ed*:

- 5020 **HOME** This variable shall determine the pathname of the user's
 5021 home directory.
- 5022 **LANG** This variable shall determine the locale to use for the
 5023 locale categories when both **LC_ALL** and the correspond-
 5024 ing environment variable (beginning with **LC_**) do not
 5025 specify a locale. See 2.6.
- 5026 **LC_ALL** This variable shall determine the locale to be used to over-
 5027 ride any values for locale categories specified by the set-
 5028 tings of **LANG** or any environment variables beginning
 5029 with **LC_**.
- 5030 **LC_COLLATE** This variable shall determine the locale for the behavior of
 5031 ranges, equivalence classes, and multicharacter collating
 5032 elements within regular expressions.

5033 **LC_CTYPE** This variable shall determine the locale for the interpreta-
 5034 tion of sequences of bytes of text data as characters (e.g.,
 5035 single- versus multibyte characters in arguments and
 5036 input files), the behavior of character classes within regu-
 5037 lar expressions.

5038 **LC_MESSAGES** This variable shall determine the language in which mes-
 5039 sages should be written.

5040 **4.20.5.4 Asynchronous Events**

5041 The `ed` utility shall take the standard action for all signals (see 2.11.5.4), with the
 5042 following exceptions:

5043 **SIGINT** The `ed` utility shall interrupt its current activity, write the string
 5044 "?\n"
 5045 to standard output, and return to command mode (see 4.20.7).

5046 **SIGHUP** If the buffer is not empty and has changed since the last write,
 5047 the `ed` utility shall attempt to write a copy of the buffer in a file.
 5048 First, the file named `ed.hup` in the current directory shall be
 5049 used; if that fails, the file named `ed.hup` in the directory named
 5050 by the **HOME** environment variable shall be used. In any case,
 5051 the `ed` utility shall exit without returning to command mode.

5052 **4.20.6 External Effects**

5053 **4.20.6.1 Standard Output**

5054 Various editing commands and the prompting feature (see `-p`) write to standard
 5055 output, as described in 4.20.7.

5056 **4.20.6.2 Standard Error**

5057 Used only for diagnostic messages.

5058 **4.20.6.3 Output Files**

5059 The output files shall be text files whose formats are dependent on the editing
 5060 commands given.

4.20.7 Extended Description

The `ed` utility shall operate on a copy of the file it is editing; changes made to the copy shall have no effect on the file until a `w` (write) command is given. The copy of the text is called the *buffer* in this clause, although no attempt is made to imply a specific implementation.

Commands to `ed` have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses very often can be omitted. If the `-p` option is specified, the prompt string shall be written to standard output before each command is read.

In general, only one command can appear on a line. Certain commands allow text to be input. This text is placed in the appropriate place in the buffer. While `ed` is accepting text, it is said to be in *input mode*. In this mode, no commands shall be recognized; all input is merely collected. Input mode is terminated by entering a line consisting of two characters: a period (.) followed by a `<newline>`. This line is not considered part of the input text.

4.20.7.1 `ed` Regular Expressions

The `ed` utility shall support basic regular expressions, as described in 2.8.3. Since regular expressions in `ed` are always matched against single lines, never against any larger section of text, there is no way for a regular expression to match a `<newline>`. A null RE shall be equivalent to the last RE encountered.

Regular expressions are used in addresses to specify lines, and in some commands (for example, the `s` substitute command) to specify portions of a line to be substituted.

4.20.7.2 `ed` Addresses

Addressing in `ed` relates to the *current line*. Generally, the current line is the last line affected by a command. The *current line number* is the address (line number) of the current line. The exact effect on the current line number is discussed under the description of each command. The `f`, `h`, `H`, `k`, `P`, `w`, `=`, and `!` commands shall not modify the current line number.

Addresses are constructed as follows:

- (1) The character `.` (period) shall address the current line.
- (2) The character `$` shall address the last line of the buffer.
- (3) A positive decimal number *n* shall address the *n*-th line of the buffer. The first line in the buffer is line number 1.
- (4) `'x` shall address the line marked with the mark name character *x*, which shall be a lowercase letter from the portable character set. Lines can be marked with the `k` command described in 4.20.7.3.13.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

- 5100 (5) An RE enclosed by slashes (/) shall address the first line found by search-
 5101 ing forward from the line following the current line toward the end of the
 5102 buffer and stopping at the first line containing a string matching the RE.
 5103 [As stated in 4.20.7.1, an address consisting of a null RE delimited by
 5104 slashes (/ /) shall address the next line containing the last RE encoun-
 5105 tered.] If necessary, the search shall wrap around to the beginning of the
 5106 buffer and continue up to and including the current line, so that the
 5107 entire buffer is searched. Within the RE, the sequence \ / shall represent
 5108 a literal slash instead of the RE delimiter.
- 5109 (6) An RE enclosed in question-marks (?) shall address the first line found by
 5110 searching backward from the line preceding the current line toward the
 5111 beginning of the buffer and stopping at the first line containing a string
 5112 matching the RE. If necessary, the search wraps around to the end of the
 5113 buffer and continues up to and including the current line. Within the RE,
 5114 the sequence \ ? shall represent a literal question-mark instead of the RE
 5115 delimiter.
- 5116 (7) An address followed by a plus sign (+) or a minus sign (-) followed by a
 5117 decimal number specifies that address plus (respectively minus) the indi-
 5118 cated number of lines. The plus sign can be omitted.
- 5119 (8) If an address begins with + or -, the addition or subtraction is taken with
 5120 respect to the current line number; for example, -5 is understood to
 5121 mean .-5.
- 5122 (9) If an address ends with + or -, then 1 shall be added to or subtracted
 5123 from the address, respectively. As a consequence of this rule and of rule
 5124 (8) immediately above, the address - shall refer to the line preceding the
 5125 current line. Moreover, trailing + and - characters shall have a cumula-
 5126 tive effect, so -- shall refer to the current line number less 2.
- 5127 (10) A comma (,) shall stand for the address pair 1, \$, while a semicolon (;)
 5128 shall stand for the pair ., \$.

5129 Commands require zero, one, or two addresses. Commands that require no
 5130 addresses shall regard the presence of an address as an error. Commands that
 5131 accept one or two addresses assume default addresses when no addresses are
 5132 given, as described in 4.20.7.3. If one address is given to a command that allows
 5133 two addresses, the command shall operate as if it were specified as:

5134 *given_address ; . command*

5135 If more addresses are given than such a command requires, the results are
 5136 undefined.

5137 Typically, addresses are separated from each other by a comma. They can also be
 5138 separated by a semicolon. In the latter case, the current line number (.) shall be
 5139 set to the first address, and only then shall the second address be calculated.
 5140 This feature can be used to determine the starting line for forward and backward
 5141 searches [see rules (5) and (6) above]. The second address of any two-address
 5142 sequence shall correspond to a line that does not precede, in the buffer, the line
 5143 corresponding to the first address.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

4.20.7.3 ed Commands

In the following list of `ed` commands, the default addresses are shown in parentheses. The number of addresses shown in the default shall be the number expected by the command. The parentheses are not part of the address; they show that the given addresses are the default.

It is generally invalid for more than one command to appear on a line. However, any command (except `e`, `E`, `f`, `q`, `Q`, `r`, `w`, and `!`) can be suffixed by the letter `l`, `n`, or `p`; in which case, except for the `l`, `n`, and `p` commands, the command shall be executed and then the new current line shall be written as described below under the `l`, `n`, and `p` commands. When an `l`, `n`, or `p` suffix is used with an `l`, `n`, or `p` command, the command shall write to standard output as described below, but it is unspecified whether the suffix writes the current line again in the requested format or whether the suffix has no effect. For example, the `pl` command (base `p` command with an `l` suffix) shall either write just the current line or shall write it twice—once as specified for `p` and once as specified for `l`. Also, the `g`, `G`, `v`, and `V` commands shall take a command as a parameter.

Each address component can be preceded by zero or more `<blank>`s. The command letter can be preceded by zero or more `<blank>`s. If a suffix letter (`l`, `n`, or `p`) is given, it shall immediately follow the command.

The `e`, `E`, `f`, `r`, and `w` commands shall take an optional *file* parameter, separated from the command letter by one or more `<blank>`s.

If changes have been made in the buffer since the last `w` command that wrote the entire buffer, `ed` shall warn the user if an attempt is made to destroy the editor buffer via the `e` or `q` commands. The `ed` utility shall write the string:

```
"?\n"
```

(followed by an explanatory message if *help mode* has been enabled via the `H` command) to standard output and shall continue in command mode with the current line number unchanged. If the `e` or `q` command is repeated with no intervening command, it shall take effect.

If an end-of-file is detected on standard input when a command is expected, the `ed` utility shall act as if a `q` command had been entered.

If the closing delimiter of an RE or of a replacement string (e.g., `/`) in a `g`, `G`, `s`, `v`, or `V` command would be the last character before a `<newline>`, that delimiter can be omitted, in which case the addressed line shall be written. For example, the following pairs of commands are equivalent:

```
s/s1/s2      s/s1/s2/p
```

```
g/s1          g/s1/p
```

```
?s1           ?s1?
```

If an invalid command is entered, `ed` shall write the string:

```
"?\n"
```


5184 (followed by an explanatory message if *help mode* has been enabled via the `H` com-
 5185 mand) to standard output and shall continue in command mode with the current
 5186 line number unchanged.

5187 **4.20.7.3.1 Append Command**

5188 *Synopsis:* (.) a
 5189 <text>
 5190 .

5191 The *append* command shall read the given text and append it after the addressed
 5192 line; the current line number shall become the address of the last inserted line,
 5193 or, if there were none, the addressed line. Address 0 shall be valid for this com-
 5194 mand: it shall cause the “appended” text to be placed at the beginning of the
 5195 buffer.

5196 **4.20.7.3.2 Change Command**

5197 *Synopsis:* (. , .) c
 5198 <text>
 5199 .

1

5200 The *change* command shall delete the addressed lines, then accept input text that
 5201 replaces these lines; the current line shall be set to the address of the last line
 5202 input; or, if there were none, at the line after the last line deleted; if the lines
 5203 deleted were originally at the end of the buffer, the current line number shall be
 5204 set to the address of the new last line; if no lines remain in the buffer, the current
 5205 line number shall be set to zero.

5206 **4.20.7.3.3 Delete Command**

5207 *Synopsis:* (. , .) d

5208 The *delete* command shall delete the addressed lines from the buffer. The address
 5209 of the line after the last line deleted shall become the current line number; if the
 5210 lines deleted were originally at the end of the buffer, the current line number
 5211 shall be set to the address of the new last line; if no lines remain in the buffer, the
 5212 current line number shall be set to zero.

5213 **4.20.7.3.4 Edit Command**

5214 *Synopsis:* e [file]

5215 The *edit* command shall delete the entire contents of the buffer and then read in
 5216 the file named by the pathname *file*. The current line number shall be set to the
 5217 address of the last line of the buffer. If no pathname is given, the currently
 5218 remembered pathname, if any, shall be used (see the `f` command). The number of
 5219 bytes read shall be written to standard output, unless the `-s` option was specified,
 5220 in the following format:

5221 "%d\n", <number of bytes read>

The name *file* shall be remembered for possible use as a default pathname in subsequent *e*, *E*, *r*, and *w* commands. If *file* is replaced by *!*, the rest of the line shall be taken to be a shell command line whose output is to be read. Such a shell command line shall not be remembered as the current *file*. All marks shall be discarded upon the completion of a successful *e* command. If the buffer has changed since the last time the entire buffer was written, the user shall be warned, as described previously.

4.20.7.3.5 Edit Without Checking Command

Synopsis: *E* [*file*]

The *Edit* command shall possess all properties and restrictions of the *e* command except that the editor shall not check to see if any changes have been made to the buffer since the last *w* command.

4.20.7.3.6 File-Name Command

Synopsis: *f* [*file*]

If *file* is given, the file-name command shall change the currently remembered pathname to *file*; whether the name is changed or not, it then shall write the (possibly new) currently remembered pathname to the standard output in the following format:

```
"%s\n", <pathname>
```

The current line number shall be unchanged.

4.20.7.3.7 Global Command

Synopsis: (*l*, *\$*)*g*/*RE*/*command list*

In the *global* command, the first step shall be to mark every line that matches the given *RE*. Then, for every such line, the given *command list* shall be executed with the current line number set to the address of that line. When the *g* command completes, the current line number shall have the value assigned by the last command in the command list. If there were no matching lines, the current line number shall not be changed. A single command or the first of a list of commands shall appear on the same line as the global command. All lines of a multiline list except the last line shall be ended with a backslash; the *a*, *i*, and *c* commands and associated input are permitted. The *.* terminating input mode can be omitted if it would be the last line of the *command list*. An empty *command list* shall be equivalent to the *p* command. The use of the *g*, *G*, *v*, *V*, and *!* commands in the *command list* produces undefined results. Any character other than *<space>* or *<newline>* can be used instead of a slash to delimit the *RE*. Within the *RE*, the *RE* delimiter itself can be used as a literal character if it is preceded by a backslash.

4.20.7.3.8 Interactive Global Command

Synopsis: (1, \$)G/*RE*/

In the *interactive global* command, the first step shall be to mark every line that matches the given *RE*. Then, for every such line, that line shall be written, the current line number shall be set to the address of that line, and any one command (other than one of the a, c, i, g, G, v, and V commands) can be input and shall be executed. A <newline> shall act as a null command (causing no action to be taken on the current line); an & shall cause the reexecution of the most recent nonnull command executed within the current invocation of G. Note that the commands input as part of the execution of the G command can address and affect any lines in the buffer. The final value of the current line number shall be the value set by the last command successfully executed. (Note that the last command successfully executed shall be the G command itself if a command fails or the null command is specified.) If there were no matching lines, the current line number shall not be changed. The G command can be terminated by a SIGINT signal. Any character other than <space> or <newline> can be used instead of a slash to delimit the *RE* and the replacement. Within the *RE*, the *RE* delimiter itself can be used as a literal character if it is preceded by a backslash.

4.20.7.3.9 Help Command

Synopsis: h

The *help* command shall write a short message to standard output that explains the reason for the most recent ? notification. The current line number shall be unchanged.

4.20.7.3.10 Help-Mode Command

Synopsis: H

The *Help* command shall cause ed to enter a mode in which help messages (see the h command) shall be written to standard output for all subsequent ? notifications. The H command alternately shall turn this mode on and off; it shall be initially off. If the help-mode is being turned on, the H command also shall explain the previous ? notification, if there was one. The current line number shall be unchanged.

4.20.7.3.11 Insert Command

Synopsis: (.)i
 <text>
 .

The *insert* command shall insert the given text before the addressed line; . shall be left at the last inserted line, or, if there was none, at the addressed line. This command differs from the a command only in the placement of the input text. Address 0 shall be invalid for this command.

4.20.7.3.12 Join Command

Synopsis: (. , .+1) j

The *join* command shall join contiguous lines by removing the appropriate <new-line> characters. If exactly one address is given, this command shall do nothing. If lines are joined, the current line number shall be set to the address of the joined line; otherwise, the current line number shall be unchanged.

4.20.7.3.13 Mark Command

Synopsis: (.) kx

The *mark* command shall mark the addressed line with name *x*, which shall be a lowercase letter from the portable character set. The address '*x*' then shall refer to this line; the current line number shall be unchanged.

4.20.7.3.14 List Command

Synopsis: (. , .) l

The *list* command shall write to standard output the addressed lines in a visually unambiguous form. The characters listed in Table 2-15 (see 2.12) shall be written as the corresponding escape sequence. Nonprintable characters not in Table 2-15 shall be written as one three-digit octal number (with a preceding <backslash>) for each byte in the character (most significant byte first). If the size of a byte on the system is greater than nine bits, the format used for nonprintable characters is implementation defined.

Long lines shall be folded, with the point of folding indicated by writing <backslash><newline>; the length at which folding occurs is unspecified, but should be appropriate for the output device. The end of each line shall be marked with a \$. An *l* command can be appended to any other command other than *e*, *E*, *f*, *q*, *Q*, *r*, *w*, or *!*. The current line number shall be set to the address of the last line written.

4.20.7.3.15 Move Command

Synopsis: (. , .) m*address*

The *move* command shall reposition the addressed line(s) after the line addressed by *address*. Address 0 shall be valid for *address* and cause the addressed line(s) to be moved to the beginning of the buffer. It shall be an error if address *address* falls within the range of moved lines. The current line number shall be set to the address of the last line moved.

4.20.7.3.16 Number Command

Synopsis: (. , .) n

The *number* command shall write to standard output the addressed lines, preceding each line by its line number and a <tab> character; the current line number shall be set to the address of the last line written. The *n* command can be

5336 appended to any other command other than `e`, `E`, `f`, `q`, `Q`, `r`, `w`, or `!`.

5337 **4.20.7.3.17 Print Command**

5338 *Synopsis:* `(. . .)p`

5339 The *print* command shall write to standard output the addressed lines; the
 5340 current line number shall be set to the address of the last line written. The `p`
 5341 command can be appended to any other command other than `e`, `E`, `f`, `q`, `Q`, `r`, `w`, or
 5342 `!`.

5343 **4.20.7.3.18 Prompt Command**

5344 *Synopsis:* `P`

5345 The *Prompt* command shall cause `ed` to prompt with an asterisk (*) (or *string*, if
 5346 `-p` is specified) for all subsequent commands. The `P` command alternately shall
 5347 turn this mode on and off; it shall be initially on if the `-p` option is specified, oth-
 5348 erwise off. The current line number shall be unchanged.

5349 **4.20.7.3.19 Quit Command**

5350 *Synopsis:* `q`

5351 The *quit* command shall cause `ed` to exit. If the buffer has changed since the last
 5352 time the entire buffer was written, the user shall be warned, as described previ-
 5353 ously.

5354 **4.20.7.3.20 Quit Without Checking Command**

5355 *Synopsis:* `Q`

5356 The *Quit* command shall cause `ed` to exit without checking if changes have been
 5357 made in the buffer since the last `w` command.

5358 **4.20.7.3.21 Read Command**

5359 *Synopsis:* `($)r [file]`

5360 The *read* command shall read in the file named by the pathname *file* and append
 5361 it after the addressed line. If no *file* argument is given, the currently remembered
 5362 pathname, if any, shall be used (see `e` and `f` commands). The currently remem-
 5363 bered pathname shall not be changed unless there is no remembered pathname.
 5364 Address 0 shall be valid for `r` and shall cause the file to be read at the beginning
 5365 of the buffer. If the read is successful, and `-s` was not specified, the number of
 5366 bytes read shall be written to standard output in the following format:

5367 `"%d\n", <number of bytes read>`

5368 The current line number shall be set to the address of the last line read in. If *file*
 5369 is replaced by `!`, the rest of the line shall be taken to be a shell command line
 5370 whose output is to be read. Such a shell command line shall not be remembered
 5371 as the current pathname.

4.20.7.3.22 Substitute Command

Synopsis: (. . .)s/*RE*/*replacement*/*flags*

The *substitute* command shall search each addressed line for an occurrence of the specified *RE* and replace either the first or all (nonoverlapped) matched strings with the *replacement*; see the following description of the *g* suffix. It is an error if the substitution fails on every addressed line. Any character other than <space> or <newline> can be used instead of a slash to delimit the *RE* and the replacement. Within the *RE*, the *RE* delimiter itself can be used as a literal character if it is preceded by a backslash. The current line shall be set to the address of the last line on which a substitution occurred.

An ampersand (&) appearing in the *replacement* shall be replaced by the string matching the *RE* on the current line. The special meaning of & in this context can be suppressed by preceding it by backslash. As a more general feature, the characters \n, where *n* is a digit, shall be replaced by the text matched by the corresponding backreference expression (see 2.8.3.3). When the character % is the only character in the *replacement*, the *replacement* used in the most recent substitute command shall be used as the *replacement* in the current substitute command; if there was no previous substitute command, the use of % in this manner shall be an error. The % shall lose its special meaning when it is in a replacement string of more than one character or is preceded by a backslash.

A line can be split by substituting a <newline> character into it. The application shall escape the <newline> in the *replacement* by preceding it by backslash. Such substitution cannot be done as part of a *g* or *v* command list. The current line number shall be set to the address of the last line on which a substitution is performed. If no substitution is performed, the current line number shall be unchanged. If a line is split, a substitution shall be considered to have been performed on each of the new lines for the purpose of determining the new current line number. A substitution shall be considered to have been performed even if the replacement string is identical to the string that it replaces.

The value of *flags* shall be zero or more of:

<i>count</i>	Substitute for the <i>count</i> th occurrence only of the <i>RE</i> found on each addressed line.
<i>g</i>	Globally substitute for all nonoverlapping instances of the <i>RE</i> rather than just the first one. If both <i>g</i> and <i>count</i> are specified, the results are unspecified.
<i>l</i>	Write to standard output the final line in which a substitution was made. The line shall be written in the format specified for the <i>l</i> command.
<i>n</i>	Write to standard output the final line in which a substitution was made. The line shall be written in the format specified for the <i>n</i> command.

5413 **p** Write to standard output the final line in which a substitution was
 5414 made. The line shall be written in the format specified for the **p** com-
 5415 mand.

5416 **4.20.7.3.23 Copy Command**

5417 *Synopsis:* (*. . .*) *t* *address*

5418 The *t* command shall be equivalent to the *m* command, except that a copy of the
 5419 addressed lines shall be placed after address *address* (which can be 0); the current
 5420 line number shall be set to the address of the last line added.

5421 **4.20.7.3.24 Undo Command**

5422 *Synopsis:* *u*

5423 The *undo* command shall nullify the effect of the most recent command that
 5424 modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*,
 5425 *u*, *v*, *G*, or *V* command. All changes made to the buffer by a *g*, *G*, *v*, or *V* global 1
 5426 command shall be “undone” as a single change; if no changes were made by the 1
 5427 global command (such as with *g/RE/p*), the *u* command shall have no effect. The 1
 5428 current line number shall be set to the value it had immediately before the com-
 5429 mand being undone started.

5430 **4.20.7.3.25 Global Non-Matched Command**

5431 *Synopsis:* (*1 , \$*) *v/RE/command list*

5432 This command shall be equivalent to the global command *g* except that the lines
 5433 that are marked during the first step shall be those that do not match the RE.

5434 **4.20.7.3.26 Interactive Global Not-Matched Command**

5435 *Synopsis:* (*1 , \$*) *V/RE/*

5436 This command shall be equivalent to the interactive global command *G* except
 5437 that the lines that are marked during the first step shall be those that do not
 5438 match the RE.

5439 **4.20.7.3.27 Write Command**

5440 *Synopsis:* (*1 , \$*) *w [file]*

5441 The *write* command shall write the addressed lines into the file named by the
 5442 pathname *file*. The command shall create the file, if it does not exist, or shall
 5443 replace the contents of the existing file. The currently remembered pathname
 5444 shall not be changed unless there is no remembered pathname. If no pathname is
 5445 given, the currently remembered pathname, if any, shall be used (see *e* and *f*
 5446 commands); the current line number shall be unchanged. If the command is suc-
 5447 cessful, the number of bytes written shall be written to standard output, unless
 5448 the *-s* option was specified, in the following format:

5449 "%d\n", <number of bytes written>
 5450 If *file* begins with **!**, the rest of the line shall be taken to be a shell command line
 5451 whose standard input shall be the addressed lines. Such a shell command line
 5452 shall not be remembered as the current pathname. This usage of the write com- 1
 5453 mand with **!** shall not be considered as a “last w command that wrote the entire 1
 5454 buffer,” as described previously; thus, this alone shall not prevent the warning to 1
 5455 the user if an attempt is made to destroy the editor buffer via the **e** or **q** com- 1
 5456 mands. 1

5457 **4.20.7.3.28 Line Number Command**

5458 *Synopsis:* (**\$**) =
 5459 The line number of the addressed line shall be written to standard output in the
 5460 following format:

5461 "%d\n", <line number>

5462 The current line number shall be unchanged by this command.

5463 **4.20.7.3.29 Shell Escape Command**

5464 *Synopsis:* **!***command*

5465 The remainder of the line after the **!** shall be sent to the command interpreter to
 5466 be interpreted as a shell command line. Within the text of that shell command
 5467 line, the unescaped character **%** shall be replaced with the remembered pathname;
 5468 if a **!** appears as the first character of the command, it shall be replaced with the
 5469 text of the previous shell command executed via **!**. Thus, **!!** shall repeat the pre- 2
 5470 vious **!command**. If any replacements of **%** and/or **!** are performed, the modified 2
 5471 line shall be written to the standard output before *command* is executed. The **!** 2
 5472 command shall write 2

5473 "**!**\n"

5474 to standard output upon completion, unless the **-s** option is specified. The
 5475 current line number shall be unchanged.

5476 **4.20.7.3.30 Null Command**

5477 *Synopsis:* (**.** + **1**)

5478 An address alone on a line shall cause the addressed line to be written. A <new-
 5479 line> alone shall be equivalent to **.+1p**. The current line number shall be set to
 5480 the address of the written line.

4.20.8 Exit Status

The `ed` utility shall exit with one of the following values:

- 0 Successful completion without any file or command errors.
- >0 An error occurred.

4.20.9 Consequences of Errors

When an error in the input script is encountered, or when an error is detected that is a consequence of the data (not) present in the file or due to an external condition such as a read or write error:

- If the standard input is a terminal device file, all input shall be flushed, and a new command read.
- If the standard input is a regular file, `ed` shall terminate with a nonzero exit status.

4.20.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

Some historical implementations contained a bug that allowed a single period to be entered in input mode as `<backslash> <period> <newline>`. This is not allowed by the POSIX.2 `ed` because there is no description of escaping any of the characters in input mode; backslashes are entered into the buffer exactly as typed. The typical method of entering a single period has been to precede it with another character and then use the substitute command to delete that character.

Because of the extremely terse nature of the default error messages, the prudent script writer will begin the `ed` input commands with an `H` command, so that if any errors do occur at least some clue as to the cause will be made available.

History of Decisions Made

The initial description of this utility was adapted from the *SVID*. It contains some features not found in Version 7 or BSD-derived systems. Some of the differences between the POSIX.2 and BSD `ed` utilities include, but need not be limited to:

- The BSD `- option` does not suppress the `!` prompt after a `!` command.
- BSD does not support the special meanings of the `%` and `!` characters within a `!` command.
- BSD does not support the *addresses* `;` and `,`.
- BSD allows the command/suffix pairs `pp`, `ll`, etc., which are unspecified in POSIX.2.
- BSD does not support the `!` character part of the `e`, `r`, or `w` commands.

- A failed `g` command in BSD sets the line number to the last line searched if there are no matches.
- BSD does not default the command list to the `p` command.
- BSD does not support the `G`, `h`, `H`, `n`, or `V` commands.
- On BSD, if there is no inserted text, the `insert` command changes the current line to the referenced line `-1`; i.e., the line before the specified line.
- On BSD, the `join` command with only a single address changes the current line to that address.
- BSD does not support the `P` command; moreover, in BSD it is synonymous with the `p` command.
- BSD does not support the *undo* of the commands `j`, `m`, `r`, `s`, or `t`.
- The BSD `ed` commands `W`, `wq`, and `z` are not present in POSIX.2.

The `-s` option was added to allow the functionality of the `-` option in a manner compatible with the Utility Syntax Guidelines. It is the intent of the working group that portable applications use the `-s` option, and that in the future the `-` option be removed from the standard.

Prior to Draft 8 there was a limit, `{ED_FILE_MAX}`, which described the historical limitations of some `ed` utilities in their handling of large files; some of these have had problems with files in the `>100KB` range. It was this limitation that prompted much of the desire to include a `split` command in the standard. Since this limit was removed, the standard requires that implementations document the file size limits imposed by `ed` in the conformance document. The limit `{ED_LINE_MAX}` was also removed; therefore, the global limit `{LINE_MAX}` is used for input and output lines.

The `\{m,n\}` notation was removed from the description of regular expressions because this functionality is now described in 2.8.3.

The manner in which the `l` command writes nonprintable characters was changed to avoid the historical backspace-overstrike method. On video display terminals, the overstrike is ambiguous because most terminals simply replace overstruck characters, making the `l` format not useful for its intended purpose of unambiguously understanding the content of the line. The historical backslash escapes were also ambiguous. (The string `"a\0011"` could represent a line containing those six characters or a line containing the three characters `'a'`, a byte with a binary value of 1, and a `'1'`.) In the format required here, a backslash appearing in the line will be written as `"\\"` so that the output is truly unambiguous. The method of marking the ends of lines was adopted from the `ex` editor (see the User Portability Extension) and is required for any line ending in `<space>s`; the `$` is placed on all lines so that a real `$` at the end of a line cannot be misinterpreted.

Systems with bytes too large to fit into three octal digits must devise other means of displaying nonprintable characters. Consideration was given to requiring that the number of octal digits be large enough to hold a byte, but this seemed to be too

5557 confusing for applications on the vast majority of systems where three digits are 1
 5558 adequate. It would be theoretically possible for the application to use the 1
 5559 `getconf` utility to find out the `{CHAR_BIT}` value and deal with such an algo- 1
 5560 rithm; however, there is really no portable way that an application can use the 1
 5561 octal values of the bytes across various coded character sets anyway, so the addi- 1
 5562 tional specification did not seem worth the effort. 1

5563 The description of how a NUL is written was removed. The NUL character cannot
 5564 be in text files, and the standard should not dictate behavior in the case of
 5565 undefined, erroneous input.

5566 The text requiring filenames accepted by the `E`, `e`, `R`, and `r` commands to be pat-
 5567 terns was removed due to balloting objections that this was undesirable and not
 5568 existing practice.

5569 The `-p` option in Drafts 8 and 9 said that it only worked when standard input was
 5570 associated with a terminal device. This has been changed to conform to existing
 5571 implementations, thereby allowing applications to interpose themselves between
 5572 a user and the `ed` utility.

5573 The form of the substitute command that uses the `n` suffix was limited to the first
 5574 512 matches in a previous draft (where this was described incorrectly as “backre-
 5575 ferencing”). This limit has been removed because there is no reason an editor pro-
 5576 cessing lines of `{LINE_MAX}` length should have this restriction. The command
 5577 `s/x/X/2047` should be able to substitute the 2047th occurrence of `x` on a line.

5578 The use of printing commands with printing suffixes (such as `pn`, `lp`, etc.) was
 5579 made unspecified because BSD-based systems allow this, whereas System V does
 5580 not.

5581 Some BSD-based systems exit immediately upon receipt of end-of-file if all of the
 5582 lines in the file had been deleted. Since POSIX.2 refers to the `q` command in this
 5583 instance, such behavior is not allowed.

5584 Some historical implementations returned exit status zero even if command
 5585 errors had occurred; this is not allowed by POSIX.2.

4.21 **env** — Set environment for command invocation

4.21.1 Synopsis

`env [-i] [name=value] ... [utility [argument ...]]`

Obsolescent Version:

`env [-] [name=value] ... [utility [argument ...]]`

4.21.2 Description

The `env` utility shall obtain the current environment, modify it according to its arguments, then invoke the utility named by the *utility* operand with the modified environment.

Optional arguments shall be passed to *utility*.

If no *utility* operand is specified, the resulting environment shall be written to the standard output, with one *name=value* pair per line.

4.21.3 Options

The `env` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except for its nonstandard usage of `-`, which is obsolescent.

The following options shall be supported by the implementation:

- `-i` Invoke *utility* with exactly the environment specified by the arguments; the inherited environment shall be ignored completely.
- `-` (Obsolescent.) Equivalent to the `-i` option.

4.21.4 Operands

The following operands shall be supported by the implementation:

- name=value* Arguments of the form *name=value* modify the execution environment, and are placed into the inherited environment before the *utility* is invoked.
- utility* The name of the utility to be invoked. If the *utility* operand names any of the special built-in utilities in 3.14, the results are undefined.
- argument* A string to pass as an argument for the invoked utility.

4.21.5 External Influences

4.21.5.1 Standard Input

None.

4.21.5.2 Input Files

None.

4.21.5.3 Environment Variables

The following environment variables shall affect the execution of `env`:

LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).
LC_MESSAGES	This variable shall determine the language in which messages should be written.
PATH	This variable shall determine the location of the <i>utility</i> , as described in 2.6. If PATH is specified as a <i>name=value</i> operand to <code>env</code> , the <i>value</i> given shall be used in the search for <i>utility</i> .

4.21.5.4 Asynchronous Events

Default.

4.21.6 External Effects

4.21.6.1 Standard Output

If no *utility* operand is specified, each *name=value* pair in the resulting environment shall be written in the form:

```
"%S=%S\n", <name>, <value>
```

5645 If the *utility* operand is specified, the `env` utility shall not write to standard out-
 5646 put.

5647 **4.21.6.2 Standard Error**

5648 Used only for diagnostic messages.

5649 **4.21.6.3 Output Files**

5650 None.

5651 **4.21.7 Extended Description**

5652 None.

5653 **4.21.8 Exit Status**

5654 If the *utility* utility is invoked, the exit status of `env` shall be the exit status of
 5655 *utility*; otherwise, the `env` utility shall exit with one of the following values:

5656	0	The <code>env</code> utility completed successfully.	
5657	1–125	An error occurred in the <code>env</code> utility.	1
5658	126	The utility specified by <i>utility</i> was found but could not be invoked.	1
5659	127	The utility specified by <i>utility</i> could not be found.	1

5660 **4.21.9 Consequences of Errors**

5661 Default.

5662 **4.21.10 Rationale.** *(This subclause is not a part of P1003.2)*

5663 **Examples, Usage**

5664 The following command:

```
5665     env -i PATH=/mybin mygrep xyz myfile
```

5666 invokes the command `mygrep` with a new **PATH** value as the only entry in its
 5667 environment. In this case, **PATH** is used to locate `mygrep`, which then must
 5668 reside in `/mybin`.

5669 As with all other utilities that invoke other utilities, the standard only specifies
 5670 what `env` does with standard input, standard output, standard error, input files,
 5671 and output files. If a utility is executed, it is not constrained by `env`'s
 5672 specification of input and output.

The `command`, `env`, `nohup`, and `xargs` utilities have been specified to use exit code 127 if an error occurs so that applications can distinguish “failure to find a utility” from “invoked utility exited with an error indication.” The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for “normal error conditions” and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with `[ENOENT]`, and uses 126 when any attempt to *exec* the utility fails for any other reason.

History of Decisions Made

The `-i` option was added to allow the functionality of the `-` option in a manner compatible with the Utility Syntax Guidelines. It is the intent of the working group that portable applications use the `-i` option, and that in the future the `-` option be removed from the standard.

Historical implementations of the `env` utility use *execvp()* or *execlp()* (see POSIX.1 {8} 3.1.2) to invoke the specified utility; this provides better performance and keeps users from having to escape characters with special meaning to the shell. Therefore, shell functions, special built-ins, and built-ins that are only provided by the shell are not found. Implementations are free to invoke a shell instead of using one of the *exec* family of routines, but if they do, they must be sure to escape any characters with special meaning to the shell so that the user does not have to be aware of the difference.

Some have suggested that `env` is redundant since the same effect is achieved by:

```
name=value ... utility [argument ...]
```

The example is equivalent to `env` when an environment variable is being added to the environment of the command, but not when the environment is being set to the given value. The `env` utility also writes out the current environment if invoked without arguments. There is sufficient functionality beyond what the example provides to justify inclusion of `env`.

5704 **4.22 `expr` — Evaluate arguments as an expression**

5705 **4.22.1 Synopsis**

5706 `expr operand ...`

5707 **4.22.2 Description**

5708 The `expr` utility shall evaluate an expression and write the result to standard
5709 output.

5710 **4.22.3 Options**

5711 None.

5712 **4.22.4 Operands**

5713 The single expression evaluated by `expr` shall be formed from the operands, as
5714 described in 4.22.7. Each of the expression operator symbols:

5715 `() | & = > >= < <= != + - * / % :`

5716 and the symbols *integer* and *string* in the table shall be provided by the applica-
5717 tion as separate arguments to `expr`.

5718 **4.22.5 External Influences**

5719 **4.22.5.1 Standard Input**

5720 None.

5721 **4.22.5.2 Input Files**

5722 None.

5723 **4.22.5.3 Environment Variables**

5724 The following environment variables shall affect the execution of `expr`:

5725	LANG	This variable shall determine the locale to use for the
5726		locale categories when both LC_ALL and the correspond-
5727		ing environment variable (beginning with LC_) do not
5728		specify a locale. See 2.6.

5729	LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
5730		
5731		
5732		
5733	LC_COLLATE	This variable shall determine the locale for the behavior of ranges, equivalence classes, and multicharacter collating elements within regular expressions and by the string comparison operators.
5734		
5735		
5736		
5737	LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments) and the behavior of character classes within regular expressions.
5738		
5739		
5740		
5741	LC_MESSAGES	This variable shall determine the language in which messages should be written.
5742		

5743 **4.22.5.4 Asynchronous Events**

5744 Default.

5745 **4.22.6 External Effects**

5746 **4.22.6.1 Standard Output**

5747 The *expr* utility shall evaluate the expression and write the result to standard
 5748 output. The character '0' shall be written to indicate a zero value and nothing
 5749 shall be written to indicate a null string.

5750 **4.22.6.2 Standard Error**

5751 Used only for diagnostic messages.

5752 **4.22.6.3 Output Files**

5753 None.

5754 **4.22.7 Extended Description**

5755 The formation of the expression to be evaluated is shown in Table 4-7. The sym-
 5756 bols *expr*, *expr1*, and *expr2* represent expressions formed from *integer* and *string*
 5757 symbols and the expression operator symbols (all separate arguments) by recur-
 5758 sive application of the constructs described in the table. The expressions in
 5759 Table 4-7 are listed in order of increasing precedence, with equal-precedence
 5760 operators grouped between horizontal lines. All of the operators shall be left-
 5761 associative.

Table 4-7 – *expr* Expressions

Expression	Description
<i>expr1</i> <i>expr2</i>	Returns the evaluation of <i>expr1</i> if it is neither null nor zero; otherwise, returns the evaluation of <i>expr2</i> .
<i>expr1</i> & <i>expr2</i>	Returns the evaluation of <i>expr1</i> if neither expression evaluates to null or zero; otherwise, returns zero.
<i>expr1</i> = <i>expr2</i> <i>expr1</i> > <i>expr2</i> <i>expr1</i> >= <i>expr2</i> <i>expr1</i> < <i>expr2</i> <i>expr1</i> <= <i>expr2</i> <i>expr1</i> != <i>expr2</i>	Returns the result of a decimal integer comparison if both arguments are integers; otherwise, returns the result of a string comparison using the locale-specific collation sequence. The result of each comparison shall be 1 if the specified relation is true, or 0 if the relation is false. Equal. Greater than. Greater than or equal. Less than. Less than or equal. Not equal.
<i>expr1</i> + <i>expr2</i> <i>expr1</i> - <i>expr2</i>	Addition of decimal integer-valued arguments. Subtraction of decimal integer-valued arguments.
<i>expr1</i> * <i>expr2</i> <i>expr1</i> / <i>expr2</i> <i>expr1</i> % <i>expr2</i>	Multiplication of decimal integer-valued arguments. Integer division of decimal integer-valued arguments, producing an integer result. Remainder of integer division of decimal integer-valued arguments.
<i>expr1</i> : <i>expr2</i>	Matching expression. See 4.22.7.1.
(<i>expr</i>)	Grouping symbols. Any expression can be placed within parentheses. Parentheses can be nested to a depth of {EXPR_NEST_MAX}.
<i>integer</i> <i>string</i>	An argument consisting only of an (optional) unary minus followed by digits. A string argument. See 4.22.7.2.

4.22.7.1 Matching Expression

The ' : ' matching operator shall compare the string resulting from the evaluation of *expr1* with the regular expression pattern resulting from the evaluation of *expr2*. Regular expression syntax shall be that defined in 2.8.3 (Basic Regular Expressions), except that all patterns are “anchored” to the beginning of the string (that is, only sequences starting at the first character of a string shall be matched by the regular expression) and, therefore, it is unspecified whether ^ is a special character in that context. Usually, the matching operator shall return a string representing the number of characters matched ("0" on failure). Alternatively, if the pattern contains at least one regular expression subexpression [\ (... \)], the string corresponding to \1 shall be returned (see 2.8.3.3).

5802 **4.22.7.2 String Operand**

5803 A string argument is an argument that cannot be identified as an *integer* argu-
5804 ment or as one of the expression operator symbols shown in 4.22.4.

5805 The use of string arguments `length`, `substr`, `index`, or `match` produces
5806 unspecified results.

5807 **4.22.8 Exit Status**

5808 The `expr` utility shall exit with one of the following values:

- 5809 0 If the *expression* evaluates to neither null nor zero.
- 5810 1 If the *expression* evaluates to null or zero.
- 5811 2 For invalid *expressions*.
- 5812 >2 An error occurred.

5813 **4.22.9 Consequences of Errors**

5814 Default.

5815 **4.22.10 Rationale.** (*This subclause is not a part of P1003.2*)

5816 **Examples, Usage**

5817 The `expr` utility has a rather difficult syntax:

- 5818 — Many of the operators are also shell control operators or reserved words, so
5819 they have to be escaped on the command line.
- 5820 — Each part of the expression is composed of separate arguments, so liberal
5821 usage of `<blank>`s is required. For example:

	Invalid	Valid
5823	<code>expr 1+2</code>	<code>expr 1 + 2</code>
5824	<code>expr "1 + 2"</code>	<code>expr 1 + 2</code>
5825	<code>expr 1 + (2 * 3)</code>	<code>expr 1 + \(2 * 3 \)</code>

5826 In many cases, the arithmetic and string features provided as part of the shell
5827 command language are easier to use than their equivalents in `expr`; the utility
5828 was retained by POSIX.2 as acknowledgment of the many historical shell scripts
5829 that use it. Newly written scripts should avoid `expr` in favor of the new features
5830 within the shell.

5831 The following command

5832 `a=$(expr $a + 1)`

5833 adds 1 to the variable **a**. A new application should use

1

5834 a=\$((\$a+1))

5835 The following command, for \$a equal to either /usr/abc/file or just file:

5836 expr \$a : '.*\/\(.*\)' \| \$a

5837 returns the last segment of a pathname (i.e., file). Applications should avoid
5838 the character / used alone as an argument: expr may interpret it as the division
5839 operator.

5840 The following command:

5841 expr "//\$a" : '.*\/\(.*\)'

5842 is a better representation of the previous example. The addition of the // charac-
5843 ters eliminates any ambiguity about the division operator and simplifies the
5844 whole expression. Also note that pathnames may contain characters contained in
5845 the IFS variable and should be quoted to avoid having \$a expand into multiple
5846 arguments.

5847 The following command

5848 expr "\$VAR" : '.*'

5849 returns the number of characters in VAR.

5850 **Usage Warning:** After argument processing by the shell, expr is not required to
5851 be able to tell the difference between an operator and an operand except by the
5852 value. If \$a is =, the command:

5853 expr \$a = '='

5854 looks like:

5855 expr = = =

5856 as the arguments are passed to expr (and they all may be taken as the = opera-
5857 tor). The following works reliably:

5858 expr X\$a = X=

5859 Also note that this standard permits implementations to extend utilities. The
5860 expr utility permits the integer arguments to be preceded with a unary minus.
5861 This means that an integer argument could look like an option. Therefore, the
5862 portable application must employ the "--" construct of Guideline 10 (see 2.10.2)
5863 to protect its operands if there is any chance the first operand might be a negative
5864 integer (or any string with a leading minus).

5865 History of Decisions Made

5866 In an earlier draft, Extended Regular Expressions were used in the matching
5867 expression syntax. This was changed to the Basic variety to avoid breaking his-
5868 torical applications.

5869 The use of a leading circumflex in the regular expression is unspecified because
5870 many historical implementations have treated it as special, despite their system
5871 documentation. For example,

5872 `expr foo : ^foo` `expr ^foo : ^foo`

5873 return 3 and 0, respectively, on those systems; their documentation would imply
 5874 the reverse. Thus, the anchoring condition is left unspecified to avoid breaking
 5875 historical scripts relying on this undocumented feature.

5876 **4.23 false — Return false value**

5877 **4.23.1 Synopsis**

5878 `false`

5879 **4.23.2 Description**

5880 The `false` utility shall return with a nonzero exit code.

5881 **4.23.3 Options**

5882 None.

5883 **4.23.4 Operands**

5884 None.

5885 **4.23.5 External Influences**

5886 **4.23.5.1 Standard Input**

5887 None.

5888 **4.23.5.2 Input Files**

5889 None.

5890 **4.23.5.3 Environment Variables**

5891 None.

5892 **4.23.5.4 Asynchronous Events**

5893 Default.

5894 **4.23.6 External Effects**

5895 **4.23.6.1 Standard Output**

5896 None.

5897 **4.23.6.2 Standard Error**

5898 None.

5899 **4.23.6.3 Output Files**

5900 None.

5901 **4.23.7 Extended Description**

5902 None.

5903 **4.23.8 Exit Status**

5904 The `false` utility always shall exit with a value other than zero.

5905 **4.23.9 Consequences of Errors**

5906 Default.

5907 **4.23.10 Rationale.** *(This subclause is not a part of P1003.2)*

5908 **Examples, Usage**

5909 The `false` utility is typically used in shell control structures like `while`.

4.24 find — Find files

4.24.1 Synopsis

`find path ... [operand_expression ...]`

4.24.2 Description

The `find` utility shall recursively descend the directory hierarchy from each file specified by *path*, evaluating a Boolean expression composed of the primaries described in 4.24.4 for each file encountered.

The `find` utility shall be able to descend to arbitrary depths in a file hierarchy and shall not fail due to path length limitations (unless a path operand specified by the application exceeds {PATH_MAX} requirements).

The `find` utility requires that the underlying system provides information equivalent to the *st_dev*, *st_mode*, *st_nlink*, *st_uid*, *st_gid*, *st_size*, *st_atime*, *st_mtime*, and *st_ctime* members of *struct stat* described by POSIX.1 {8} 5.6 and conforming to the *file times update* definition in 2.2.2.69.

4.24.3 Options

None.

4.24.4 Operands

The following operands shall be supported by the implementation:

The *path* operand is a pathname of a starting point in the directory hierarchy.

The first argument that starts with a `-`, or is a `!` or a `(`, and all subsequent arguments shall be interpreted as an *expression* made up of the following primaries and operators. In the descriptions, wherever *n* is used as a primary argument, it shall be interpreted as a decimal integer optionally preceded by a plus (+) or minus (−) sign, as follows:

`+n` More than *n*

`n` Exactly *n*

`−n` Less than *n*

Implementations shall recognize the following primaries: *Editor's Note: These primaries have been sorted alphabetically, without diff marks.*

`−atime n` The primary shall evaluate as true if the file access time subtracted from the initialization time is *n*−1 to *n* multiples of 24 hours. The initialization time shall be a time between the invocation of the `find` utility and the first

5943		access by that invocation of the <code>find</code> utility to any file
5944		specified by its <i>path</i> operands.
5945	<code>-ctime <i>n</i></code>	The primary shall evaluate as true if the time of last
5946		change of file status information subtracted from the ini-
5947		tialization time is $n-1$ to n multiples of 24 hours. The ini-
5948		tialization time shall be a time between the invocation of
5949		the <code>find</code> utility and the first access by that invocation of
5950		the <code>find</code> utility to any file specified by its <i>path</i> operands.
5951	<code>-depth</code>	The primary always shall evaluate as true; it shall cause
5952		descent of the directory hierarchy to be done so that all
5953		entries in a directory are acted on before the directory
5954		itself. If a <code>-depth</code> primary is not specified, all entries in a
5955		directory shall be acted on after the directory itself. If any
5956		<code>-depth</code> primary is specified, it shall apply to the entire
5957		expression even if the <code>-depth</code> primary would not nor-
5958		normally be evaluated.
5959	<code>-exec <i>utility_name</i> [<i>argument</i> ...] ;</code>	
5960		The primary shall evaluate as true if the invoked utility
5961		<i>utility_name</i> returns a zero value as exit status. The end
5962		of the primary expression shall be punctuated by a semi-
5963		colon. A <i>utility_name</i> or <i>argument</i> containing only the
5964		two characters <code>{}</code> shall be replaced by the current path-
5965		name. If a <i>utility_name</i> or <i>argument</i> string contains the
5966		two characters <code>{}</code> , but not just the two characters <code>{}</code> , it is
5967		implementation defined whether <code>find</code> replaces those two
5968		characters with the current pathname or uses the string
5969		without change. The current directory for the invocation
5970		of <i>utility_name</i> shall be the same as the current directory
5971		when the <code>find</code> utility was started. If the <i>utility_name</i>
5972		names any of the special built-in utilities in 3.14, the
5973		results are undefined.
5974	<code>-group <i>gname</i></code>	The primary shall evaluate as true if the file belongs to
5975		the group <i>gname</i> . If <i>gname</i> is a decimal integer and the
5976		<code>getgrnam()</code> (or equivalent) function does not return a
5977		valid group name, <i>gname</i> shall be interpreted as a group
5978		ID.
5979	<code>-links <i>n</i></code>	The primary shall evaluate as true if the file has n links.
5980	<code>-mtime <i>n</i></code>	The primary shall evaluate as true if the file modification
5981		time subtracted from the initialization time is $n-1$ to n
5982		multiples of 24 hours. The initialization time shall be a
5983		time between the invocation of the <code>find</code> utility and the
5984		first access by that invocation of the <code>find</code> utility to any
5985		file specified by its <i>path</i> operands.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

5986	<code>-name <i>pattern</i></code>	The primary shall evaluate as true if the basename of the filename being examined matches <i>pattern</i> using the pattern matching notation described in 3.13.
5987		
5988		
5989	<code>-newer <i>file</i></code>	The primary shall evaluate as true if the modification time of the current file is more recent than the modification time of the file named by the pathname <i>file</i> .
5990		
5991		
5992	<code>-nogroup</code>	The primary shall evaluate as true if the file belongs to a group ID for which the POSIX.1 {8} <i>getgrgid()</i> (or equivalent) function returns NULL .
5993		
5994		
5995	<code>-nouser</code>	The primary shall evaluate as true if the file belongs to a user ID for which the POSIX.1 {8} <i>getpwuid()</i> (or equivalent) function returns NULL .
5996		
5997		
5998	<code>-ok <i>utility_name</i> [<i>argument ...</i>] ;</code>	
5999		The <code>-ok</code> primary shall be equivalent to <code>-exec</code> , except that
6000		<code>find</code> shall request affirmation of the invocation of
6001		<i>utility_name</i> using the current file as an argument by
6002		writing to standard error as, described in 4.24.6.2. If the
6003		response on standard input is affirmative, the utility shall
6004		be invoked. Otherwise, the command shall not be invoked
6005		and the value of the <code>-ok</code> operand shall be false.
6006	<code>-perm [-]<i>mode</i></code>	The <i>mode</i> argument is used to represent file mode bits. It
6007		shall be identical in format to the <i>symbolic_mode</i> operand
6008		described in 4.7, and shall be interpreted as follows. To
6009		start, a template shall be assumed with all file mode bits
6010		cleared. An <i>op</i> symbol of <code>+</code> shall set the appropriate mode
6011		bits in the template; <code>-</code> shall clear the appropriate bits; <code>=</code>
6012		shall set the appropriate mode bits, without regard to the
6013		contents of process's file mode creation mask. The <i>op</i> sym-
6014		bol of <code>-</code> cannot be the first character of <i>mode</i> .
6015		If the hyphen is omitted, the primary shall evaluate as
6016		true when the file permission bits exactly match the value
6017		of the resulting template.
6018		Otherwise, if <i>mode</i> is prefixed by a hyphen, the primary
6019		shall evaluate as true if at least all the bits in the result-
6020		ing template are set in the file permission bits.
6021	<code>-perm [-]<i>onum</i></code>	(Obsolescent.) If the hyphen is omitted, the primary shall
6022		evaluate as true when the file permission bits exactly
6023		match the value of the octal number <i>onum</i> and only the
6024		bits corresponding to the octal mask 07777 shall be com-
6025		pared. (See the description of the octal <i>mode</i> in 4.7.) Oth-
6026		erwise, if <i>onum</i> is prefixed by a hyphen, the primary shall
6027		evaluate as true if at least all of the bits specified in <i>onum</i>
6028		that are also set in the octal mask 07777 are set.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

6029	<code>-print</code>	The primary always shall evaluate as true; it shall cause
6030		the current pathname to be written to standard output.
6031	<code>-prune</code>	The primary always shall evaluate as true; it shall cause
6032		<code>find</code> not to descend the current pathname if it is a direc-
6033		tory. If the <code>-depth</code> primary is specified, the <code>-prune</code> pri-
6034		mary shall have no effect.
6035	<code>-size n[c]</code>	The primary shall evaluate as true if the file size in bytes,
6036		divided by 512 and rounded up to the next integer, is <i>n</i> . If
6037		<i>n</i> is followed by the character <i>c</i> , the size shall be in bytes.
6038	<code>-type c</code>	The primary shall evaluate as true if the type of the file is
6039		<i>c</i> , where <i>c</i> is <i>b</i> , <i>c</i> , <i>d</i> , <i>p</i> , or <i>f</i> for block special file, character
6040		special file, directory, FIFO, or regular file, respectively.
6041	<code>-user uname</code>	The primary shall evaluate as true if the file belongs to
6042		the user <i>uname</i> . If <i>uname</i> is a decimal integer and the
6043		<code>getpwnam()</code> (or equivalent) function does not return a
6044		valid user name, <i>uname</i> shall be interpreted as a user ID.
6045	<code>-xdev</code>	The primary always shall evaluate as true; it shall cause
6046		<code>find</code> not to continue descending past directories that
6047		have a different device ID (<i>st_dev</i> , see POSIX.1 {8} 5.6.2).
6048		If any <code>-xdev</code> primary is specified, it shall apply to the
6049		entire expression even if the <code>-xdev</code> primary would not
6050		normally be evaluated.
6051	The primaries can be combined using the following operators (in order of decreas-	
6052	ing precedence):	
6053	<code>(expression)</code>	True if <i>expression</i> is true.
6054	<code>! expression</code>	Negation of a primary; the unary NOT operator.
6055	<code>expression [-a] expression</code>	
6056		Conjunction of primaries; the AND operator shall be
6057		implied by the juxtaposition of two primaries or made
6058		explicit by the optional <code>-a</code> operator. The second expres-
6059		sion shall not be evaluated if the first expression is false.
6060	<code>expression -o expression</code>	
6061		Alternation of primaries; the OR operator. The second
6062		expression shall not be evaluated if the first expression is
6063		true.
6064	If no <i>expression</i> is present, <code>-print</code> shall be used as the expression. Otherwise, if	
6065	the given expression does not contain any of the primaries <code>-exec</code> , <code>-ok</code> , or	
6066	<code>-print</code> , the given expression shall be effectively replaced by:	
6067	<code>(given_expression) -print</code>	
6068	The <code>-user</code> , <code>-group</code> , and <code>-newer</code> primaries each shall evaluate their respective	
6069	arguments only once.	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

6070 4.24.5 External Influences

6071 4.24.5.1 Standard Input

6072 If the `-ok` primary is used, the response shall be read from the standard input.
 6073 An entire line shall be read as the response. Otherwise, the standard input shall
 6074 not be used.

6075 4.24.5.2 Input Files

6076 None.

6077 4.24.5.3 Environment Variables

6078 The following environment variables shall affect the execution of `find`:

6079	LANG	This variable shall determine the locale to use for the
6080		locale categories when both LC_ALL and the correspond-
6081		ing environment variable (beginning with LC_) do not
6082		specify a locale. See 2.6.
6083	LC_ALL	This variable shall determine the locale to be used to over-
6084		ride any values for locale categories specified by the set-
6085		tings of LANG or any environment variables beginning
6086		with LC_ .
6087	LC_COLLATE	This variable shall determine the locale for the behavior of
6088		ranges, equivalence classes, and multicharacter collating
6089		elements used in the pattern matching notation for the
6090		<code>-name</code> option and in the extended regular expression
6091		defined for the <code>yesexpr</code> locale keyword in the
6092		LC_MESSAGES category.
6093	LC_CTYPE	This variable shall determine the locale for the interpreta-
6094		tion of sequences of bytes of text data as characters (e.g.,
6095		single- versus multibyte characters in arguments), the
6096		behavior of character classes within the pattern matching
6097		notation used for the <code>-name</code> option, and the behavior of
6098		character classes within regular expressions used in the
6099		extended regular expression defined for the <code>yesexpr</code>
6100		locale keyword in the LC_MESSAGES category.
6101	LC_MESSAGES	This variable shall determine the processing of affirmative
6102		responses and the language in which messages should be
6103		written.
6104	PATH	This variable shall determine the location of the
6105		<i>utility_name</i> for the <code>-exec</code> and <code>-ok</code> primaries, as
6106		described in 2.6.

6107 **4.24.5.4 Asynchronous Events**

6108 Default.

6109 **4.24.6 External Effects**

6110 **4.24.6.1 Standard Output**

6111 The `-print` primary shall cause the current pathnames to be written to standard
6112 output. The format shall be:

6113 "`%s\n`", *<path>*

6114 **4.24.6.2 Standard Error**

6115 The `-ok` primary shall write a prompt to standard error containing at least the
6116 `utility_name` to be invoked and the current pathname. In the POSIX Locale, the
6117 last non-`<blank>` character in the prompt shall be `?`. The exact format used is
6118 unspecified.

6119 Otherwise, the standard error shall be used only for diagnostic messages.

6120 **4.24.6.3 Output Files**

6121 None.

6122 **4.24.7 Extended Description**

6123 None.

6124 **4.24.8 Exit Status**

6125 The `find` utility shall exit with one of the following values:

6126 0 All *path* operands were traversed successfully.

6127 >0 An error occurred.

6128 **4.24.9 Consequences of Errors**

6129 Default.

6130 **4.24.10 Rationale.** *(This subclause is not a part of P1003.2)*

6131 **Examples, Usage**

6132 When used in operands, pattern matching notation, semicolons, opening
6133 parentheses, and closing parentheses are special to the shell and must be quoted
6134 (see 3.2).

6135 The following command:

```
6136     find / \( -name tmp -o -name '*.xx' \) \  
6137         -atime +7 -exec rm {} \;
```

6138 removes all files named `tmp` or ending in `.xx` that have not been accessed for
6139 seven or more 24-hour periods.

6140 The following command:

```
6141     find . -perm -o+w,+s
```

6142 prints (`-print` is assumed) the names of all files in or below the current direc-
6143 tory, with all of the file permission bits `S_ISUID`, `S_ISGID`, and `S_IWOTH` set.

6144 The `-prune` primary was adopted from later releases of 4.3BSD and the third edi- 1
6145 tion of the *SVID*. The following command recursively prints pathnames of all files 1
6146 in the current directory and below, but skips directories named `SCCS` and files in
6147 them.

```
6148     find . -name SCCS -prune -o -print
```

6149 The following command behaves as in the previous example, but prints the names
6150 of the `SCCS` directories.

```
6151     find . -print -name SCCS -prune
```

6152 The following command is roughly equivalent to the `-nt` extension to `test`: 1

```
6153     if [ -n "$(find file1 -prune -newer file2)" ]; then 2  
6154         printf %s\\n "file1 is newer than file2" 2  
6155     fi 1
```

6156 **History of Decisions Made**

6157 The historical `-a` operator is kept as an optional operator for compatibility with
6158 existing shell scripts even though it is redundant with expression concatenation.

6159 The symbolic means of specifying file permission bits, based on `chmod`, was added
6160 in response to numerous balloting objections that `find` was the only remaining
6161 utility to not support this method. The warning about a leading *Op* of `-` is to
6162 avoid ambiguity with the optional leading hyphen. Since the initial mode is all
6163 bits off, there are not any symbolic modes that need to use `-` as the first character.
6164 The bit that is traditionally used for sticky (historically 01000) is still specified in
6165 the `-perm` primary using the octal number argument form. Since this bit is not
6166 defined by POSIX.1 {8} or POSIX.2, applications must not assume that it actually
6167 refers to the traditional sticky bit.

The descriptions of how the `-` modifier on the *mode* and *onum* arguments to the `-perm` primary affects processing has been documented here to match the way it behaves in practice on historical BSD and System V implementations. System V and BSD documentation both describe it in terms of checking additional bits; in fact, it uses the same bits, but checks for having at least all of the matching bits set instead of having exactly the matching bits set.

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified, because:

- (1) Implementations may desire more descriptive prompts than those used on historical implementations.
- (2) Since the traditional prompt strings do not terminate with `<newline>`s, there is no portable way for another program to interact with the prompts of this utility via pipes.

Therefore, an application using this prompting option relies on the system to provide the most suitable dialogue directly with the user, based on the general guidelines specified.

The `-name file` operand was changed to use the shell pattern matching notation so that `find` is consistent with other utilities using pattern matching.

For the `-type c` operand, implementors of symbolic links should consider `l` (the letter ell) for symbolic links. Implementations that support sockets also use `-type s` for sockets. Implementations planning to add options to allow `find` to follow symbolic links or treat them as special files, should consider using `-follow` as used in BSD and System V Release 4 as a guide.

The `-size` operand refers to the size of a file, rather than the number of blocks it may occupy in the file system. The intent is that the POSIX.1 {8} *st_size* field should be used, not the *st_blocks* found in historical implementations. There are at least two reasons for this:

- In both System V and BSD, `find` only uses *st_size* in size calculations for the operands specified by POSIX.2. (BSD uses *st_blocks* only when processing the `-ls` primary.)
- Users will usually be thinking of size in terms of the size of the file in bytes, which is also used by the `ls` utility for the output from the `-l` option. (In both System V in BSD, `ls` uses *st_size* for the `-l` option size field and uses *st_blocks* for the `ls -s` calculations. POSIX.2 does not specify `ls -s`.)

The descriptions of `-atime`, `-ctime`, and `-mtime` were changed from the *SVID*'s description of *n* “days” to “24-hour periods.” For example, a file accessed at 23:59 will be selected by

```
find . -atime -1 -print
```

at 00:01 the next day (less than 24 hours later, not more than one day ago); the midnight boundary between days has no effect on the 24-hour calculation. The description is also different in terms of the exact timeframe for the *n* case (versus the *+n* or *-n*), but it matches all known historical implementations. It refers to

6210 one 24-hour period in the past, not any time from the beginning of that period to 1
 6211 the current time. For example, `-atime 3` is true if the file was accessed any time 1
 6212 in the period from 72 to 48 hours ago. 1

6213 Historical implementations do not modify `{ }` when it appears as a substring of an
 6214 `-exec` or `-ok utility_name` or argument string. There have been numerous user
 6215 requests for this extension, so this standard allows the desired behavior. At least
 6216 one recent implementation does support this feature, but ran into several prob-
 6217 lems in managing memory allocation and dealing with multiple occurrences of `{ }`
 6218 in a string while it was being developed, so it is not yet required behavior.

6219 Assuming the presence of `-print` was added at the request of several working
 6220 group members to correct a historical pitfall that plagues novice users. It is
 6221 entirely upward compatible from the historical System V `find` utility and should
 6222 be easy to implement. In its simplest form (`find directory`), it could be confused
 6223 with the historical BSD fast `find`. The BSD developers agree that adding `-print`
 6224 as a default expression is the right thing to do and believe that the fast `find`
 6225 functionality should have been/should be provided by a separate utility. They
 6226 suggest that the new utility be called `locate`.

6227 4.25 `fold` — Fold lines

6228 4.25.1 Synopsis

6229 `fold [-bs] [-w width] [file ...]`

6230 4.25.2 Description

6231 The `fold` utility is a filter that shall fold lines from its input files, breaking the
 6232 lines to have a maximum of *width* column positions (or bytes, if the `-b` option is
 6233 specified). Lines shall be broken by the insertion of a `<newline>` character such
 6234 that each output line (referred to later in this clause as a segment) is the max-
 6235 imum width possible that does not exceed the specified number of column posi-
 6236 tions (or bytes). A line shall not be broken in the middle of a character. The
 6237 behavior is undefined if *width* is less than the number of columns any single char-
 6238 acter in the input would occupy.

6239 If the `<carriage-return>`, `<backspace>`, or `<tab>` characters are encountered 2
 6240 in the input, and the `-b` option is not specified, they shall be treated specially:

6241 `<carriage-return>` 2
 6242 The current count of line width shall be set to zero. The `fold` utility 2
 6243 shall not insert a `<newline>` immediately before or after any 2
 6244 `<carriage-return>`. 2

6245 `<backspace>`
 6246 The current count of line width shall be decremented by one, although
 6247 the count never shall become negative. The `fold` utility shall not

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

6248 insert a <newline> immediately before or after any <backspace>.

6249 <tab> Each <tab> character encountered shall advance the column position
6250 pointer to the next tab stop. Tab stops shall be at each column position
6251 *n* such that *n* modulo 8 equals 1.

6252 **4.25.3 Options**

6253 The `fold` utility shall conform to the utility argument syntax guidelines
6254 described in 2.10.2.

6255 The following options shall be supported by the implementation:

6256 **-b** Count *width* in bytes rather than column positions.

6257 **-s** If a segment of a line contains a <blank> within the first *width*
6258 column positions (or bytes), break the line after the last such
6259 <blank> meeting the width constraints. If there is no <blank>
6260 meeting the requirements, the **-s** option shall have no effect for
6261 that output segment of the input line.

6262 **-w *width*** Specify the maximum line length, in column positions (or bytes if
6263 **-b** is specified). The results are unspecified if *width* is not a posi-
6264 tive decimal number. The default value shall be 80.

6265 **4.25.4 Operands**

6266 The following operand shall be supported by the implementation:

6267 *file* A pathname of a text file to be folded. If no *file* operands are
6268 specified, the standard input shall be used.

6269 **4.25.5 External Influences**

6270 **4.25.5.1 Standard Input**

6271 The standard input shall be used only if no *file* operands are specified. See Input
6272 Files.

6273 **4.25.5.2 Input Files**

6274 If the **-b** option is specified, the input files shall be text files except that the lines
6275 are not limited to {LINE_MAX} bytes in length. If the **-b** option is not specified,
6276 the input files shall be text files.

4.25.5.3 Environment Variables

The following environment variables shall affect the execution of `fold`:

LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files) and for the determination of the width in column positions each character would occupy on a constant-width-font output device.
LC_MESSAGES	This variable shall determine the language in which messages should be written.

4.25.5.4 Asynchronous Events

Default.

4.25.6 External Effects

4.25.6.1 Standard Output

The standard output shall be a file containing a sequence of characters whose order shall be preserved from the input file(s), possibly with inserted `<newline>` characters.

4.25.6.2 Standard Error

Used only for diagnostic messages.

4.25.6.3 Output Files

None.

4.25.7 Extended Description

None.

4.25.8 Exit Status

The `fold` utility shall exit with one of the following values:

0 All input files were processed successfully.

>0 An error occurred.

4.25.9 Consequences of Errors

Default.

4.25.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The `cut` and `fold` utilities can be used to create text files out of files with arbitrary line lengths. The `cut` utility should be used when the number of lines (or records) needs to remain constant. The `fold` utility should be used when the contents of long lines needs to be kept contiguous.

The `fold` utility is frequently used to send text files to line printers that truncate, rather than fold, lines wider than the printer is able to print (usually 80 or 132 column positions.)

Although terminal input in canonical processing mode requires the erase character (frequently set to `<backspace>`) to erase the previous character (not byte or column position), terminal output is not buffered and is extremely difficult, if not impossible, to parse correctly; the interpretation depends entirely on the physical device that will actually display/print/store the output. In all known internationalized implementations, the utilities producing output for mixed column width output assume that a `<backspace>` backs up one column position and outputs enough `<backspace>`s to get back to the start of the character when `<backspace>` is used to provide local line motions to support underlining and emboldening operations. Since `fold` without the `-b` option is dealing with these same constraints, `<backspace>` is always treated as backing up one column position rather than backing up one character.

An example invocation that submits a file of possibly long lines to the line printer (under the assumption that the user knows the line width of the printer to be assigned by `lp`):

```
fold -w 132 bigfile | lp
```

History of Decisions Made

Historical versions of the `fold` utility assumed one byte was one character and occupied one column position when written out. This is no longer always true. Since the most common usage of `fold` is believed to be folding long lines for output to limited-length output devices, this capability was preserved as the default case. The `-b` option was added so that applications could `fold` files with arbitrary length lines into text files that could then be processed by the utilities in this standard. Note that although the width for the `-b` option is in bytes, a line will never be split in the middle of a character. (It is unspecified what happens if a width is specified that is too small to hold a single character found in the input followed by a `<newline>`.)

The use of a hyphen as an option to specify standard input was removed from an earlier draft because it adds no functionality and is not historical practice.

The tab stops are hardcoded to be every eighth column to meet historical practice. No new method of specifying other tab stops was invented.

4.26 `getconf` — Get configuration values

4.26.1 Synopsis

`getconf system_var`

`getconf path_var pathname`

4.26.2 Description

In the first synopsis form, the `getconf` utility shall write to the standard output the value of the variable specified by the *system_var* operand.

In the second synopsis form, the `getconf` utility shall write to the standard output the value of the variable specified by the *path_var* operand for the path specified by the *pathname* operand.

The value of each configuration variable shall be determined as if it were obtained by calling the function from which it is defined to be available by this standard or by POSIX.1 {8} (see Operands). The value shall reflect conditions in the current operating environment.

4.26.3 Options

None.

4.26.4 Operands

The following operands shall be supported by the implementation:

system_var A name of a configuration variable whose value is available from the function defined in 7.8.1 [such as *confstr()* in the C binding], from the POSIX.1 {8} *sysconf()* function, one of the additional POSIX.2 variables described in 7.8.2, to be available from the *sysconf()* function, or a minimum value specified by POSIX.1 {8} or POSIX.2 for one of these variables.

The configuration variables and minimum values listed in the:

- Name column of Table 2-16 (Utility Limit Minimum Values)
- Name column of Table 2-17 (Symbolic Utility Limits)
- Name column of Table 2-18 (Optional Facility Configuration Values)
- Name column of POSIX.1 {8} Table 2-3 (Minimum Values)
- Name column of POSIX.1 {8} Table 2-4 (Run-Time Increaseable Values)
- Variable column of POSIX.1 {8} Table 4-2 (Configurable System Variables; except CLK_TCK need not be supported), without the enclosing braces and PATH [corresponding to the *confstr()* name value _CS_PATH] shall be recognized as valid *system_var* operands. The implementation may support additional *system_var* operand values.

path_var A name of a configuration variable whose value is available from the POSIX.1 {8} *pathconf()* function.

The configuration variables listed in the Variable column of the POSIX.1 {8} Table 5-2 (Configurable Pathname Variables), without the enclosing braces, shall be recognized as valid *path_var* operands. The implementation may support additional *path_var* operand values.

pathname A pathname for which the variable specified by *path_var* is to be determined.

6401 **4.26.5 External Influences**

6402 **4.26.5.1 Standard Input**

6403 None.

6404 **4.26.5.2 Input Files**

6405 None.

6406 **4.26.5.3 Environment Variables**

6407 The following environment variables shall affect the execution of `getconf`:

6408	LANG	This variable shall determine the locale to use for the
6409		locale categories when both LC_ALL and the correspond-
6410		ing environment variable (beginning with LC_) do not
6411		specify a locale. See 2.6.

6412	LC_ALL	This variable shall determine the locale to be used to over-
6413		ride any values for locale categories specified by the set-
6414		tings of LANG or any environment variables beginning
6415		with LC_ .

6416	LC_CTYPE	This variable shall determine the locale for the interpreta-
6417		tion of sequences of bytes of text data as characters (e.g.,
6418		single- versus multibyte characters in arguments).

6419	LC_MESSAGES	This variable shall determine the language in which mes-
6420		sages should be written.

6421 **4.26.5.4 Asynchronous Events**

6422 Default.

6423 **4.26.6 External Effects**

6424 **4.26.6.1 Standard Output**

6425 If the specified variable is defined on the system and its value is described to be
 6426 available from the function in 7.8.1, its value shall be written in the following for-
 6427 mat:

6428 "**%s\n**", *<value>*

6429 Otherwise, if the specified variable is defined on the system, its value shall be
 6430 written in the following format:

6431 "**%d\n**", *<value>*

6432 If the specified variable is valid, but is undefined on the system, `getconf` shall
6433 write using the following format:

6434 "undefined\n"

6435 If the variable name is invalid or an error occurs, nothing shall be written to stan-
6436 dard output.

6437 **4.26.6.2 Standard Error**

6438 Used only for diagnostic messages.

6439 **4.26.6.3 Output Files**

6440 None.

6441 **4.26.7 Extended Description**

6442 None.

6443 **4.26.8 Exit Status**

6444 The `getconf` utility shall exit with one of the following values:

6445 0 The specified variable is valid and information about its current state
6446 was written successfully.

6447 >0 An error occurred.

6448 **4.26.9 Consequences of Errors**

6449 Default.

6450 **4.26.10 Rationale.** *(This subclause is not a part of P1003.2)*

6451 **Examples, Usage**

6452 The original need for this utility, and for the `confstr()` function, was to provide a
6453 way of finding the configuration-defined default value for the **PATH** environment
6454 variable. Since **PATH** can be modified by the user to include directories that
6455 could contain utilities replacing the POSIX.2 standard utilities, shell scripts need
6456 a way to determine the system supplied **PATH** environment variable value that
6457 contains the correct search path for the standard utilities.

6458 It was later suggested that access to the other variables described here could also
6459 be useful to applications.

6460 This example illustrates the value of `{NGROUPS_MAX}`:

```
6461         getconf NGROUPS_MAX
```

6462 This example illustrates the value of {NAME_MAX} for a specific directory:

```
6463         getconf NAME_MAX /usr
```

6464 This example shows how to deal more carefully with results that might be
6465 unspecified:

```
6466         if value=$(getconf PATH_MAX /usr); then
6467             if [ "$value" = "undefined" ]; then
6468                 echo PATH_MAX in /usr is infinite.
6469             else
6470                 echo PATH_MAX in /usr is $value.
6471             fi
6472         else
6473             echo Error in getconf.
6474         fi
```

1

6475 Note that:

```
6476         sysconf(_SC_POSIX_C_BIND);
```

6477 and:

```
6478         system("getconf POSIX2_C_BIND");
```

6479 in a C program could give different answers. The *sysconf()* call supplies a value
6480 that corresponds to the conditions when the program was either compiled or exe-
6481 cuted, depending on the implementation; the *system()* call to *getconf* always
6482 supplies a value corresponding to conditions when the program is executed.

6483 History of Decisions Made

6484 This utility was renamed from *posixconf* during balloting because the new
6485 name expresses its purpose more specifically, and does not unduly restrict the
6486 scope of application of the utility.

6487 This functionality of this utility would not be adequately subsumed by another
6488 command such as

```
6489         grep var /etc/conf
```

6490 because such a strategy would provide correct values for neither those variables
6491 that can vary at run-time, nor those that can vary depending on the path.

6492 Previous versions of this utility specified exit status 1 when the specified variable
6493 was valid, but not defined on the system. The output string "undefined" is now
6494 used to specify this case with exit code 0 because so many things depend on an
6495 exit code of zero when an invoked utility is successful.

4.27 getopt — Parse utility options

4.27.1 Synopsis

`getopts` *optstring name* [*arg ...*]

4.27.2 Description

The `getopts` utility can be used to retrieve options and option-arguments from a list of parameters. It shall support the utility argument syntax guidelines 3 through 10, inclusive, described in 2.10.2.

Each time it is invoked, the `getopts` utility shall place the value of the next option in the shell variable specified by the *name* operand and the index of the next argument to be processed in the shell variable **OPTIND**. Whenever the shell is invoked, **OPTIND** shall be initialized to 1.

When the option requires an option-argument, the `getopts` utility shall place it in the shell variable **OPTARG**. If no option was found, or if the option that was found does not have an option-argument, **OPTARG** shall be unset.

1

If an option character not contained in the *optstring* operand is found where an option character is expected, the shell variable specified by *name* shall be set to the question-mark (?) character. In this case, if the first character in *optstring* is a colon (:), the shell variable **OPTARG** shall be set to the option character found, but no output shall be written to standard error; otherwise, the shell variable **OPTARG** shall be unset and a diagnostic message shall be written to standard error. This condition shall be considered to be an error detected in the way arguments were presented to the invoking application, but shall not be an error in `getopts` processing.

If an option-argument is missing:

- If the first character of *optstring* is a colon, the shell variable specified by *name* shall be set to the colon character and the shell variable **OPTARG** shall be set to the option character found.

- Otherwise, the shell variable specified by *name* shall be set to the question-mark character, the shell variable **OPTARG** shall be unset, and a diagnostic message shall be written to standard error. This condition shall be considered to be an error detected in the way arguments were presented to the invoking application, but shall not be an error in `getopts` processing; a diagnostic message shall be written as stated, but the exit status shall be zero.

When the end of options is encountered, the `getopts` utility shall exit with a return value greater than zero; the shell variable **OPTIND** shall be set to the index of the first nonoption-argument, where the first `--` argument is considered to be an option-argument if there are no other nonoption-arguments appearing before it, or the value `$# + 1` if there are no nonoption-arguments; the *name*

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

variable shall be set to the question-mark character. Any of the following shall identify the end of options: the special option `--`, finding an argument that does not begin with a `-`, or encountering an error.

The shell variables **OPTIND** and **OPTARG** shall be local to the caller of `getopts` and shall not be exported by default.

The shell variable specified by the *name* operand, **OPTIND**, and **OPTARG** shall affect the current shell execution environment; see 3.12.

If the application sets **OPTIND** to the value 1, a new set of parameters can be used: either the current positional parameters or new *arg* values. Any other attempt to invoke `getopts` multiple times in a single shell execution environment with parameters (positional parameters or *arg* operands) that are not the same in all invocations, or with an **OPTIND** value modified to be a value other than 1, produces unspecified results.

4.27.3 Options

None.

4.27.4 Operands

The following operands shall be supported by the implementation:

<p><i>optstring</i></p>	<p>A string containing the option characters recognized by the utility invoking <code>getopts</code>. If a character is followed by a colon, the option shall be expected to have an argument, which should be supplied as a separate argument. Applications should specify an option character and its option-argument as separate arguments, but <code>getopts</code> shall interpret the characters following an option character requiring arguments as an argument whether or not this is done. An explicit null option-argument need not be recognized if it is not supplied as a separate argument when <code>getopts</code> is invoked. [See also the <i>getopt()</i> Description in B.7]. The characters question-mark and colon shall not be used as option characters by an application. The use of other option characters that are not alphanumeric produces unspecified results. If the option-argument is not supplied as a separate argument from the option character, the value in OPTARG shall be stripped of the option character and the <code>'-'</code>. The first character in <i>optstring</i> shall determine how <code>getopts</code> shall behave if an option character is not known or an option-argument is missing. See 4.27.2.</p>
-------------------------	--

<p><i>name</i></p>	<p>The name of a shell variable that shall be set by the <code>getopts</code> utility to the option character that was found. See 4.27.2.</p>
--------------------	---

The `getopts` utility by default shall parse positional parameters passed to the invoking shell procedure. If *args* are given, they shall be parsed instead of the positional parameters.

6575 4.27.5 External Influences

6576 4.27.5.1 Standard Input

6577 None.

6578 4.27.5.2 Input Files

6579 None.

6580 4.27.5.3 Environment Variables

6581 The following environment variables shall affect the execution of `getopts`:

6582	LANG	This variable shall determine the locale to use for the
6583		locale categories when both LC_ALL and the correspond-
6584		ing environment variable (beginning with LC_) do not
6585		specify a locale. See 2.6.
6586	LC_ALL	This variable shall determine the locale to be used to over-
6587		ride any values for locale categories specified by the set-
6588		tings of LANG or any environment variables beginning
6589		with LC_ .
6590	LC_CTYPE	This variable shall determine the locale for the interpreta-
6591		tion of sequences of bytes of text data as characters (e.g.,
6592		single- versus multibyte characters in arguments).
6593	LC_MESSAGES	This variable shall determine the language in which mes-
6594		sages should be written.
6595	OPTIND	This variable shall be used by the <code>getopts</code> utility as the
6596		index of the next argument to be processed.

6597 4.27.5.4 Asynchronous Events

6598 Default.

6599 4.27.6 External Effects

6600 4.27.6.1 Standard Output

6601 None.

6602 4.27.6.2 Standard Error

6603 Whenever an error is detected and the first character in the *optstring* operand is
 6604 not a colon (:), a diagnostic message shall be written to standard error with the 1
 6605 following information in an unspecified format: 1

- 6606 — The invoking program name shall be identified in the message. The invoking 1
 6607 program name shall be the value of the shell special parameter 0 (see 1
 6608 3.5.2) at the time the `getopts` utility is invoked. A name equivalent to 1
 6609 `basename "$0"` 1
 6610 may be used. 1
 6611 — If an option is found that was not specified in *optstring*, this error shall be 1
 6612 identified and the invalid option character shall be identified in the mes- 1
 6613 sage. 1
 6614 — If an option requiring an option-argument is found, but an option-argument 1
 6615 is not found, this error shall be identified and the invalid option character 1
 6616 shall be identified in the message. 1

6617 **4.27.6.3 Output Files**

6618 None.

6619 **4.27.7 Extended Description**

6620 None.

6621 **4.27.8 Exit Status**

6622 The `getopts` utility shall exit with one of the following values:

- 6623 0 An option, specified or unspecified by *optstring*, was found.
 6624 >0 The end of options was encountered or an error occurred.

6625 **4.27.9 Consequences of Errors**

6626 Default.

6627 **4.27.10 Rationale.** (*This subclause is not a part of P1003.2*)

6628 **Examples, Usage**

6629 The `getopts` utility was chosen in preference to the `getopt` utility specified in
 6630 System V because `getopts` handles option-arguments containing <blank> char-
 6631 acters.

6632 Since `getopts` affects the current shell execution environment, it is generally
 6633 provided as a shell regular built-in. If it is called in a subshell or separate utility 1
 6634 execution environment, such as one of the following: 1

```

6635         (getopts abc value "$@")
6636         nohup getopts ...
6637         find . -exec getopts ... \;

```

6638 it will not affect the shell variables in the caller's environment. 1

6639 Note that shell functions share **OPTIND** with the calling shell even though the
6640 positional parameters are changed. Functions that want to use `getopts` to parse
6641 their arguments will usually want to save the value of **OPTIND** on entry and
6642 restore it before returning. However, there will be cases when a function will
6643 want to change **OPTIND** for the calling shell.

6644 The following example script parses and displays its arguments:

```

6645 aflag=
6646 bflag=
6647 while getopts ab: name
6648 do
6649     case $name in
6650         a)      aflag=1;;
6651         b)      bflag=1
6652                bval="$OPTARG";
6653         ?)      printf "Usage: %s: [-a] [-b value] args\n" $0
6654                exit 2;;
6655     esac
6656 done
6657 if [ ! -z "$aflag" ]; then
6658     printf "Option -a specified\n"
6659 fi
6660 if [ ! -z "$bflag" ]; then
6661     printf 'Option -b "%s" specified\n' "$bval"
6662 fi
6663 shift $((OPTIND - 1))
6664 printf "Remaining arguments are: %s\n" "$*"

```

6665 History of Decisions Made

6666 The **OPTARG** variable is not mentioned in the Environment Variables subclause
6667 because it does not affect the execution of `getopts`; it is one of the few “output-
6668 only” variables used by the standard utilities.

6669 Use of colon (:) as an option character (in a previous draft) was new behavior and
6670 violated the syntax guidelines. Many objectors felt that it did not add enough to
6671 `getopts` to warrant mandating the extension to existing practice. The colon is
6672 now specified to behave as in the KornShell version of the `getopts` utility; when
6673 used as the first character in the *optstring* operand, it disables diagnostics con-
6674 cerning missing option-arguments and unexpected option characters. This
6675 replaces the use of the **OPTERR** variable that was specified in an earlier draft.

6676 The formats of the diagnostic messages produced by the `getopts` utility and the
6677 `getopt()` function are not fully specified because implementations with superior
6678 (“friendlier”) formats objected to the formats used by some historical implementa-
6679 tions. It was felt to be important that the information in the messages used be 1

uniform between `getopts` and `getopt()`. Exact duplication of the messages might not be possible, particularly if a utility is built on another system that has a different `getopt()` function, but the messages must have specific information included so that the program name, invalid option character, and type of error can be distinguished by a user.

Only a rare application program will intercept a `getopts` standard error message and want to parse it. Therefore, implementations are free to choose the most usable messages they can devise. The following formats are used by many historical implementations:

```
%s: illegal option -- %c\n", <program name>,
<option character>
```

```
%s: option requires an argument -- %c\n",
<program name>, <option character>
```

Historical shells with built-in versions of `getopt()` or `getopts` have used different formats, frequently not even indicating the option character found in error.

4.28 grep — File pattern searcher

4.28.1 Synopsis

```
grep [-E | -F] [-c | -l | -q] [-insvx] -e pattern_list ... [-f pattern_file]
... [file ...]
```

```
grep [-E | -F] [-c | -l | -q] [-insvx] [-e pattern_list] ... -f pattern_file
... [file ...]
```

```
grep [-E | -F] [-c | -l | -q] [-insvx] pattern_list [file ...]
```

Obsolescent Versions:

```
egrep [-c | -l] [-inv] -e pattern_list [file ...]
```

```
egrep [-c | -l] [-inv] -f pattern_file [file ...]
```

```
egrep [-c | -l] [-inv] pattern_list [file ...]
```

```
fgrep [-c | -l] [-invx] -e pattern_list [file ...]
```

```
fgrep [-c | -l] [-invx] -f pattern_file [file ...]
```

```
fgrep [-c | -l] [-invx] pattern_list [file ...]
```

4.28.2 Description

The `grep` utility shall search the input files, selecting lines matching one or more patterns; the types of patterns shall be controlled by the options specified. The patterns are specified by the `-e` option, `-f` option, or the *pattern_list* operand. The *pattern_list*'s value shall consist of one or more patterns separated by `<newline>`s; the *pattern_file*'s contents shall consist of one or more patterns terminated by `<newline>`s. By default, an input line shall be selected if any pattern, treated as an entire basic regular expression (BRE) as described in 2.8.3, matches any part of the line; a null BRE shall match every line. By default, each selected input line shall be written to the standard output.

Regular expression matching shall be based on text lines. Since `<newline>` separates or terminates patterns (see the `-e` and `-f` options below), regular expressions cannot contain a `<newline>` character. Similarly, since patterns are matched against individual lines of the input, there is no way for a pattern to match a `<newline>` found in the input.

A command invoking the (obsolescent) `egrep` utility with the `-e` option specified shall be equivalent to the command:

```
grep -E [-c | -l] [-inv] -e pattern_list [file ...]
```

A command invoking the `egrep` utility with the `-f` option specified shall be equivalent to the command:

```
grep -E [-c | -l] [-inv] -f pattern_file [file ...]
```

A command invoking the `egrep` utility with the *pattern_list* specified shall be equivalent to the command:

```
grep -E [-c | -l] [-inv] pattern_list [file ...]
```

A command invoking the (obsolescent) `fgrep` utility with the `-e` option specified shall be equivalent to the command:

```
grep -F [-c | -l] [-invx] -e pattern_list [file ...]
```

A command invoking the `fgrep` utility with the `-f` option specified shall be equivalent to the command:

```
grep -F [-c | -l] [-invx] -f pattern_file [file ...]
```

A command invoking the `fgrep` utility with the *pattern_list* operand specified shall be equivalent to the command:

```
grep -F [-c | -l] [-invx] pattern_list [file ...]
```

4.28.3 Options

The `grep` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

- `-E` Match using extended regular expressions. Treat each pattern specified as an ERE, as described in 2.8.4. If any entire ERE pattern matches an input line, the line shall be matched. A null ERE shall match every line.
- `-F` Match using fixed strings. Treat each pattern specified as a string instead of a regular expression. If an input line contains any of the patterns as a contiguous sequence of bytes, the line shall be matched. A null string shall match every line.
- `-c` Write only a count of selected lines to standard output.
- `-e pattern_list` Specify one or more patterns to be used during the search for input. Patterns in *pattern_list* shall be separated by a <newline>. A null pattern can be specified by two adjacent <newline>s in *pattern_list*; in the obsolescent forms, adjacent <newline>s in *pattern_list* produce undefined results. Unless the `-E` or `-F` option is also specified, each pattern shall be treated as a BRE, as described in 2.8.3. In the nonobsolescent forms, multiple `-e` and `-f` options shall be accepted by the `grep` utility. All of the specified patterns shall be used when matching lines, but the order of evaluation is unspecified.
- `-f pattern_file` Read one or more patterns from the file named by the pathname *pattern_file*. Patterns in *pattern_file* shall be terminated by a <newline>. A null pattern can be specified by an empty line in *pattern_file*. Unless the `-E` or `-F` option is also specified, each pattern shall be treated as a BRE, as described in 2.8.3.
- `-i` Perform pattern matching in searches without regard to case. See 2.8.2.
- `-l` (The letter ell.) Write only the names of files containing selected lines to standard output. Pathnames shall be written once per file searched. If the standard input is searched, a pathname of "(standard input)" shall be written, in the POSIX Locale. In other locales, standard input may be replaced by something more appropriate in those locales.
- `-n` Precede each output line by its relative line number in the file, each file starting at line 1. The line number counter shall be reset for each file processed.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

6783 -q Quiet. Do not write anything to the standard output, regardless
 6784 of matching lines. Exit with zero status if an input line is
 6785 selected.

6786 -s Suppress the error messages ordinarily written for nonexistent or
 6787 unreadable files. Other error messages shall not be suppressed.

6788 -v Select lines not matching any of the specified patterns. If the -v
 6789 option is not specified, selected lines shall be those that match
 6790 any of the specified patterns.

6791 -x Consider only input lines that use all characters in the line to
 6792 match an entire fixed string or regular expression to be matching
 6793 lines.

6794 **4.28.4 Operands**

6795 The following operands shall be supported by the implementation:

6796 *pattern* Specify one or more patterns to be used during the search for
 6797 input. This operand shall be treated as if it were specified as
 6798 -e *pattern_list* (see 4.28.3).

6799 *file* A pathname of a file to be searched for the pattern(s). If no *file*
 6800 operands are specified, the standard input shall be used.

6801 **4.28.5 External Influences**

6802 **4.28.5.1 Standard Input**

6803 The standard input shall be used only if no *file* operands are specified. See Input
 6804 Files.

6805 **4.28.5.2 Input Files**

6806 The input files shall be text files.

6807 **4.28.5.3 Environment Variables**

6808 The following environment variables shall affect the execution of *grep*:

6809 **LANG** This variable shall determine the locale to use for the
 6810 locale categories when both **LC_ALL** and the correspond-
 6811 ing environment variable (beginning with **LC_**) do not
 6812 specify a locale. See 2.6.

6813 **LC_ALL** This variable shall determine the locale to be used to over-
 6814 ride any values for locale categories specified by the set-
 6815 tings of **LANG** or any environment variables beginning
 6816 with **LC_**.

6817	LC_COLLATE	This variable shall determine the locale for the behavior of
6818		ranges, equivalence classes, and multicharacter collating
6819		elements within regular expressions.
6820	LC_CTYPE	This variable shall determine the locale for the interpreta-
6821		tion of sequences of bytes of text data as characters (e.g.,
6822		single- versus multibyte characters in arguments) and the
6823		behavior of character classes within regular expressions.
6824	LC_MESSAGES	This variable shall determine the language in which mes-
6825		sages should be written.

6826 **4.28.5.4 Asynchronous Events**

6827 Default.

6828 **4.28.6 External Effects**

6829 **4.28.6.1 Standard Output**

6830 If the `-l` option is in effect, and the `-q` option is not, a single output line shall be
6831 written for each file containing at least one selected input line:

6832 `"%s\n", file`

6833 Otherwise, if more than one *file* argument appears, and `-q` is not specified, the
6834 `grep` utility shall prefix each output line by:

6835 `"%s: ", file`

6836 The remainder of each output line shall depend on the other options specified:

6837 — If the `-c` option is in effect, the remainder of each output line shall contain:

6838 `"%d\n", <count>`

6839 — Otherwise, if `-c` is not in effect and the `-n` option is in effect, the following
6840 shall be written to standard output:

6841 `"%d: ", <line number>`

6842 — Finally, the following shall be written to standard output:

6843 `"%s ", <selected-line contents>`

6844 **4.28.6.2 Standard Error**

6845 Used only for diagnostic messages.

4.28.6.3 Output Files

None.

4.28.7 Extended Description

None.

4.28.8 Exit Status

The `grep` utility shall exit with one of the following values:

- 0 One or more lines were selected.
- 1 No lines were selected.
- >1 An error occurred.

4.28.9 Consequences of Errors

If the `-q` option is specified, the exit status shall be zero if an input line is selected, even if an error was detected. Otherwise, default actions shall be performed.

4.28.10 Rationale. *(This subclause is not a part of P1003.2)***Examples, Usage**

This `grep` has been enhanced in an upward-compatible way to provide the exact functionality of the historical `egrep` and `fgrep` commands as well. It was the clear intention of the working group to consolidate the three greps into a single command.

The old `egrep` and `fgrep` commands are likely to be supported for many years to come as implementation extensions, allowing existing applications to operate unmodified. 1

To find all uses of the word `Posix` (in any case) in the file `text.mm`, and write with line numbers:

```
grep -i -n posix text.mm
```

To find all empty lines in the standard input: 2

```
grep ^$
or
grep -v .
```

Both of the following commands print all lines containing strings `abc` or `def` or both:

```
6877      grep -E 'abc
6878      def'
```

```
6879      grep -F 'abc
6880      def'
```

6881 Both of the following commands print all lines matching exactly `abc` or `def`:

```
6882      grep -E '^abc$
6883      ^def$'
```

```
6884      grep -F -x 'abc
6885      def'
```

6886 History of Decisions Made

6887 The `-e pattern_list` option has the same effect as the *pattern_list* operand, but is
 6888 useful when *pattern_list* begins with the hyphen delimiter. It is also useful when
 6889 it is more convenient to provide multiple patterns as separate arguments.

6890 Earlier drafts did not show that the `-c`, `-l`, and `-q` options were mutually
 6891 exclusive. This has been fixed to more closely align with historical practice and
 6892 documentation.

6893 Historical implementations usually silently ignored all but one of multiply
 6894 specified `-e` and `-f` options, but were not consistent as to which specification was
 6895 actually used.

6896 POSIX.2 requires that the nonobsolescent forms accept multiple `-e` and `-f` options
 6897 and use all of the patterns specified while matching input text lines. [Note that
 6898 the order of evaluation is not specified. If an implementation finds a null string
 6899 as a pattern, it is allowed to use that pattern first (matching every line) and effec-
 6900 tively ignore any other patterns.]

6901 The `-b` option was removed from the Options subclause, since block numbers are
 6902 implementation dependent.

6903 The System V restriction on using `-` to mean standard input was lifted.

6904 A definition of action taken when given a null RE or ERE is specified. This is an
 6905 error condition in some historical implementations.

6906 The `-l` option previously indicated that its use was undefined when no files were
 6907 explicitly named. This behavior was historical and placed an unnecessary restric-
 6908 tion on future implementations. It has been removed.

6909 The `-q` option was added at the suggestion of members of the balloting group as a
 6910 means of easily determining whether or not a pattern (or string) exists in a group
 6911 of files. When searching several files, it provides a performance improvement
 6912 (because it can quit as soon as it finds the first match) and requires less care by
 6913 the user in choosing the set of files to supply as arguments (because it will exit
 6914 zero if it finds a match even if `grep` detected an access or read error on earlier file
 6915 operands).

6916 The historical BSD `grep -s` option practice is easily duplicated by redirecting
 6917 standard output to `/dev/null`. The `-s` option required here is from System V.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

6918 The `-x` option, historically available only with `fgrep`, is available here for all of
 6919 the nonobsolescent versions.

6920 **4.29 head — Copy the first part of files**

6921 **4.29.1 Synopsis**

6922 `head [-n number] [file ...]`

6923 *Obsolescent version:*

6924 `head [-number] [file ...]`

6925 **4.29.2 Description**

6926 The `head` utility shall copy its input files to the standard output, ending the out-
 6927 put for each file at a designated point.

6928 Copying shall end at the point in each input file indicated by the `-n number`
 6929 option (or the obsolescent version's `-number` argument). The option-argument
 6930 *number* shall be counted in units of lines.

6931 **4.29.3 Options**

6932 The `head` utility shall conform to the utility argument syntax guidelines
 6933 described in standard described in 2.10.2, except that the obsolescent version
 6934 accepts multicharacter numeric options.

6935 The following option shall be supported by the implementation in the nonobsoles-
 6936 cent version:

6937 `-n number` The first *number* lines of each input file shall be copied to stan-
 6938 dard output. The *number* option argument shall be a positive
 6939 decimal integer.

6940 If no options are specified, `head` shall act as if `-n 10` had been specified.

6941 In the obsolescent version, the following option shall be supported by the imple-
 6942 mentation:

6943 `-number` The *number* argument is a positive decimal integer with the same
 6944 effect as the `-nnumber` option in the nonobsolescent version.

4.29.4 Operands

The following operand shall be supported by the implementation:

<p>6947 <i>file</i></p> <p>6948</p>	<p>A pathname of an input file. If no <i>file</i> operands are specified, the standard input shall be used.</p>
---	---

4.29.5 External Influences

4.29.5.1 Standard Input

The standard input shall be used only if no *file* operands are specified. See Input Files.

4.29.5.2 Input Files

Input files shall be text files, but the line length shall not be restricted to {LINE_MAX} bytes.

4.29.5.3 Environment Variables

The following environment variables shall affect the execution of head:

<p>6958 LANG</p> <p>6959</p> <p>6960</p> <p>6961</p>	<p>This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.</p>
<p>6962 LC_ALL</p> <p>6963</p> <p>6964</p> <p>6965</p>	<p>This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_.</p>
<p>6966 LC_CTYPE</p> <p>6967</p> <p>6968</p> <p>6969</p>	<p>This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).</p>
<p>6970 LC_MESSAGES</p> <p>6971</p>	<p>This variable shall determine the language in which messages should be written.</p>

4.29.5.4 Asynchronous Events

Default.

4.29.6 External Effects**4.29.6.1 Standard Output**

The standard output shall contain designated portions of the input file(s).

If multiple *file* operands are specified, *head* shall precede the output for each with the header:

```
"\n==> %s <==\n", <pathname>
```

except that the first header written shall not include the initial <newline>.

4.29.6.2 Standard Error

Used only for diagnostic messages.

4.29.6.3 Output Files

None.

4.29.7 Extended Description

None.

4.29.8 Exit Status

The *head* utility shall exit with one of the following values:

0 Successful completion.

>0 An error occurred.

4.29.9 Consequences of Errors

Default.

4.29.10 Rationale. *(This subclause is not a part of P1003.2)***Usage, Examples**

The nonobsolescent version of *head* was created to allow conformance to the Utility Syntax Guidelines. The *-n* option was added to this new interface so that *head* and *tail* would be more logically related.

To write the first ten lines of all files (except those with a leading period) in the directory:

7000 head *

7001 **History of Decisions Made**

7002 The head utility was not in early drafts. It was felt that head, and its frequent
7003 companion, tail, were useful mostly to interactive users, and not application
7004 programs. However, balloting input suggested that these utilities actually do find
7005 significant use in scripts, such as to write out portions of log files. Although it is
7006 possible to simulate head with sed 10q for a single file, the working group
7007 decided that the popularity of head on historical BSD systems warranted its inclu-
7008 sion alongside tail.

7009 An earlier draft had the synopsis line:

7010 head [-c | -l] [-n *number*] [*file* ...]

7011 This was changed to the current form based on comments and objections noting
7012 that -c has not been provided by historical versions of head and other utilities in
7013 POSIX.2 provide similar functionality. Also, -l was changed to -n to match a
7014 similar change in tail.

7015 **4.30 id — Return user identity**

7016 **4.30.1 Synopsis**

7017 id [*user*]

7018 id -G [-n] [*user*]

7019 id -g [-nr] [*user*]

7020 id -u [-nr] [*user*]

7021 **4.30.2 Description**

7022 If no *user* operand is provided, the id utility shall write the user and group IDs
7023 and the corresponding user and group names of the invoking process to standard
7024 output. If the effective and real IDs do not match, both shall be written. If multi-
7025 ple groups are supported by the underlying system (see the description of
7026 {NGROUPS_MAX} in POSIX.1 {8}), the supplementary group affiliations of the
7027 invoking process also shall be written.

7028 If a *user* operand is provided and the process has the appropriate privileges, the
7029 user and group IDs of the selected user shall be written. In this case, effective IDs
7030 shall be assumed to be identical to real IDs. If the selected user has more than 1
7031 one allowable group membership listed in the group database (see POSIX.1 {8} sec- 1
7032 tion 9.1), these shall be written in the same manner as the supplementary groups 1
7033 described in the preceding paragraph. 1

4.30.3 Options

The `id` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

- `-G` Output all different group IDs (effective, real, and supplementary) only, using the format "`%u\n`". If there is more than one distinct group affiliation, output each such affiliation, using the format "`%u`", before the `<newline>` is output.
- `-g` Output only the effective group ID, using the format "`%u\n`".
- `-n` Output the name in the format "`%s`" instead of the numeric ID using the format "`%u`".
- `-r` Output the real ID instead of the effective ID.
- `-u` Output only the effective user ID, using the format "`%u\n`".

4.30.4 Operands

The following operand shall be supported by the implementation:

- `user` The login name for which information is to be written.

4.30.5 External Influences

4.30.5.1 Standard Input

None.

4.30.5.2 Input Files

None.

4.30.5.3 Environment Variables

The following environment variables shall affect the execution of `id`:

- LANG** This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.
- LC_ALL** This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

7065 **LC_CTYPE** This variable shall determine the locale for the interpreta-
 7066 tion of sequences of bytes of text data as characters (e.g.,
 7067 single- versus multibyte characters in arguments).

7068 **LC_MESSAGES** This variable shall determine the language in which mes-
 7069 sages should be written.

7070 **4.30.5.4 Asynchronous Events**

7071 Default.

7072 **4.30.6 External Effects**

7073 **4.30.6.1 Standard Output**

7074 The following formats shall be used when the LC_MESSAGES locale category
 7075 specifies the POSIX Locale. In other locales, the strings `uid`, `gid`, `euid`, `egid`,
 7076 and `groups` may be replaced with more appropriate strings corresponding to the
 7077 locale.

7078 "uid=%u(%s) gid=%u(%s)\n", *<real user ID>*, *<user-name>*,
 7079 *<real group ID>*, *<group-name>*

7080 If the effective and real user IDs do not match, the following shall be inserted
 7081 immediately before the `\n` character in the previous format:

7082 " euid=%u(%s) ",

7083 with the following arguments added at the end of the argument list:

7084 *<effective user ID>*, *<effective user-name>*

7085 If the effective and real group IDs do not match, the following shall be inserted
 7086 directly before the `\n` character in the format string (and after any addition
 7087 resulting from the effective and real user IDs not matching):

7088 " egid=%u(%s) ",

7089 with the following arguments added at the end of the argument list:

7090 *<effective group-ID>*, *<effective group name>*

7091 If the process has supplementary group affiliations or the selected user is allowed 1
 7092 to belong to multiple groups, the first shall be added directly before the *<new-* 1
 7093 *line>* character in the format string:

7094 " groups=%u(%s) "

7095 with the following arguments added at the end of the argument list:

7096 *<supplementary group ID>*, *<supplementary group name>*

7097 and the necessary number of the following added after that for any remaining
 7098 supplementary group IDs:

7099 " , %u(%s) "

7100 and the necessary number of the following arguments added at the end of the
7101 argument list:

7102 <supplementary group ID>, <supplementary group name>

7103 If any of the user ID, group ID, effective user ID, effective group ID, or 1
7104 supplementary/multiple group IDs cannot be mapped by the system into printable 1
7105 user or group names, the corresponding (%s) and name argument shall be omitted
7106 from the corresponding format string.

7107 When any of the options are specified, the output format shall be as described
7108 under 4.30.3.

7109 **4.30.6.2 Standard Error**

7110 Used only for diagnostic messages.

7111 **4.30.6.3 Output Files**

7112 None.

7113 **4.30.7 Extended Description**

7114 None.

7115 **4.30.8 Exit Status**

7116 The `id` utility shall exit with one of the following values:

7117 0 Successful completion.

7118 >0 An error occurred.

7119 **4.30.9 Consequences of Errors**

7120 Default.

7121 **4.30.10 Rationale.** *(This subclause is not a part of P1003.2)*

7122 **Examples, Usage**

7123 The functionality provided by the 4BSD `groups` utility can be simulated using:

7124 `id -Gn [user]`

7125 Note that output produced by the `-G` option and by the default case could poten-
7126 tially produce very long lines on systems that support large numbers of

supplementary groups. (On systems with user and group IDs that are 32-bit integers and with group names with a maximum of 8 bytes per name, 93 supplementary groups plus distinct effective and real group and user IDs could theoretically overflow the 2048-byte {LINE_MAX} text file line limit on the default output case. It would take about 186 supplementary groups to overflow the 2048-byte barrier using `id -G`.) This is not expected to be a problem in practice, but in cases where it is a concern, applications should consider using `fold -s` (see 4.25) before postprocessing the output of `id`.

History of Decisions Made

The 4BSD command `groups` was considered, but was not used as it did not provide the functionality of the `id` utility of the *SVID*. Also, it was thought that it would be easier to modify `id` to provide the additional functionality necessary to systems with multiple groups than to invent another command.

The options `-u`, `-g`, `-n`, and `-r` were added to ease the use of `id` with shell commands substitution. Without these options it is necessary to use some preprocessor such as `sed` to select the desired piece of information. Since output such as that produced by `id -u -n` is wanted frequently, it seemed desirable to add the options.

4.31 `join` — Relational database operator

4.31.1 Synopsis

```
join [ -a file_number | -v file_number ] [-e string] [-o list] [-t char]
      [-1 field] [-2 field] file1 file2
```

Obsolescent version:

```
join [-a file_number] [-e string] [-j field] [-j1 field] [-j2 field]
      [-o list ...] [-t char] file1 file2
```

4.31.2 Description

The `join` utility shall perform an “equality join” on the files *file1* and *file2*. The joined files shall be written to the standard output.

The “join field” is a field in each file on which the files are compared. There shall be one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line by default shall consist of the join field, then the remaining fields from *file1*, then the remaining fields from *file2*. This format can be changed by using the `-o` option (see below). The `-a` option can be used to add unmatched lines to the output. The `-v` option can be used to output only unmatched lines.

By default, the files *file1* and *file2* should be ordered in the collating sequence of `sort -b` (see 4.58) on the fields on which they are to be joined, by default the first in each line. All selected output shall be written in the same collating sequence.

The default input field separators shall be `<blank>s`. In this case, multiple separators shall count as one field separator, and leading separators shall be ignored. The default output field separator shall be a `<space>`.

The field separator and collating sequence can be changed by using the `-t` option (see below).

If the input files are not in the appropriate collating sequence, the results are unspecified.

4.31.3 Options

The `join` utility shall conform to the utility argument syntax guidelines described in 2.10.2. The obsolescent version does not follow the utility argument syntax guidelines: the `-j1` and `-j2` options are multicharacter options and the `-o` option takes multiple arguments.

The following options shall be supported by the implementation:

`-a file_number`

Produce a line for each unpairable line in file *file_number*, where *file_number* is 1 or 2, in addition to the default output. If both `-a 1` and `-a 2` are specified, all unpairable lines shall be output.

`-e string` Replace empty output fields by string *string*.

`-j field` (Obsolescent.) Equivalent to: `-1 field -2 field`

`-j1 field` (Obsolescent.) Equivalent to: `-1 field`

`-j2 field` (Obsolescent.) Equivalent to: `-2 field`

`-o list` Construct the output line to comprise the fields specified in *list*, each element of which has the form *file_number.field*, where *file_number* is a file number and *field* is a decimal integer field number. The elements of *list* are either comma- or `<blank>`-separated, as specified in Guideline 8 in 2.10.2. The fields specified by *list* shall be written for all selected output lines. Fields selected by *list* that do not appear in the input shall be treated as empty output fields. (See the `-e` option.) The join field shall not be written unless specifically requested. The *list* shall be a single command line argument. However, as an obsolescent feature, the argument *list* can be multiple arguments on the command line. If this is the case, and if the `-o` option is the last option before *file1*, and if *file1* is of the form *string.string*, the results are undefined.

- 7200 **-t *char*** Use character *char* as a separator, for both input and output.
 7201 Every appearance of *char* in a line shall be significant. When this
 7202 option is specified, the collating sequence should be the same as
 7203 sort without the **-b** option.
- 7204 **-v *file_number*** Instead of the default output, produce a line only for each unpair-
 7205 able line in *file_number*, where *file_number* is 1 or 2. If both **-v 1**
 7206 and **-v 2** are specified, all unpairable lines shall be output.
- 7208 **-1 *field*** Join on the *fieldth* field of file 1. Fields are decimal integers start-
 7209 ing with 1.
- 7210 **-2 *field*** Join on the *fieldth* field of file 2. Fields are decimal integers start-
 7211 ing with 1.

7212 **4.31.4 Operands**

7213 The following operands shall be supported by the implementation:

- 7214 *file1*
 7215 *file2* A pathname of a file to be joined. If either of the *file1* or *file2*
 7216 operands is **-**, the standard input is used in its place.

7217 **4.31.5 External Influences**

7218 **4.31.5.1 Standard Input**

7219 The standard input shall be used only if the *file1* or *file2* operand is **-**. See Input
 7220 Files.

7221 **4.31.5.2 Input Files**

7222 The input files shall be text files.

7223 **4.31.5.3 Environment Variables**

7224 The following environment variables shall affect the execution of **join**:

- 7225 **LANG** This variable shall determine the locale to use for the
 7226 locale categories when both **LC_ALL** and the correspond-
 7227 ing environment variable (beginning with **LC_**) do not
 7228 specify a locale. See 2.6.
- 7229 **LC_ALL** This variable shall determine the locale to be used to over-
 7230 ride any values for locale categories specified by the set-
 7231 tings of **LANG** or any environment variables beginning
 7232 with **LC_**.

7233	LC_COLLATE	This variable shall determine the collating sequence <code>join</code> expects to have been used when the input files were sorted.
7234		
7235		
7236	LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).
7237		
7238		
7239		
7240	LC_MESSAGES	This variable shall determine the language in which messages should be written.
7241		

7242 **4.31.5.4 Asynchronous Events**

7243 Default.

7244 **4.31.6 External Effects**

7245 **4.31.6.1 Standard Output**

7246 The `join` utility output shall be a concatenation of selected character fields.
7247 When the `-o` option is not specified, the output shall be:

7248 "*%s%s%s\n*", *<join field>*, *<other file1 fields>*, *<other file2 fields>*

7249 If the `join` field is not the first field in either file, the *<other file fields>* are:

7250 *<fields preceding join field>*, *<fields following join field>*

7251 When the `-o` option is specified, the output format shall be:

7252 "*%s\n*", *<concatenation of fields>*

7253 where the concatenation of fields is described by the `-o` option, above.

7254 For either format, each field (except the last) shall be written with its trailing
7255 separator character. If the separator is the default (*<blank>s*), a single *<space>*
7256 character shall be written after each field (except the last).

7257 **4.31.6.2 Standard Error**

7258 Used only for diagnostic messages.

7259 **4.31.6.3 Output Files**

7260 None.

7261 **4.31.7 Extended Description**

7262 None.

7263 **4.31.8 Exit Status**

7264 The `join` utility shall exit with one of the following values:

- 7265 0 All input files were output successfully.
- 7266 >0 An error occurred.

7267 **4.31.9 Consequences of Errors**

7268 Default.

7269 **4.31.10 Rationale.** *(This subclause is not a part of P1003.2)*

7270 **Examples, Usage**

7271 Pathnames consisting of numeric digits should not be specified directly following
7272 the `-o` list.

7273 The developers of the standard believed that `join` should operate as documented
7274 in the *SVID* and BSD, not as historically implemented. Historical implementa-
7275 tions do not behave as documented in these areas:

- 7276 (1) Most implementations of `join` require using the `-o` option when using
7277 the `-e` option.
- 7278 (2) Most implementations do not parse the `-o` option as documented, and
7279 parse the elements as separate *argv* items, until the item is not of the
7280 form *file_number.field*. This behavior is permitted as an obsolescent
7281 usage of the utility. To ensure maximum portability, *file1* should not be
7282 of the form *string.string*. A suitable alternative to guarantee portability
7283 would be to put the `--` flag before any *file1* operand.

7284 The obsolescent `-j`, `-j1`, and `-j2` options have been described to show how they
7285 have been used in historical implementations. Earlier drafts showed
7286 `-j file_number field`, but a space was never allowed before the *file_number* and
7287 two option arguments were never intended.

7288 **History of Decisions Made**

7289 The ability to specify *file2* as `-` is not historical practice; it was added for com-
7290 pleteness.

7291 As a result of a balloting comment, the `-v` option was added to the nonobsolescent
7292 version. This option was felt necessary because it permitted the writing of *only*
7293 those lines that do not match on the join field, as opposed to the `-a` option, which
7294 prints both lines that do and do not match. This additional facility is parallel

7295 with the `-v` option of `grep`.

7296 **4.32 kill — Terminate or signal processes**

7297 **4.32.1 Synopsis**

7298 `kill -s signal_name pid ...`

7299 `kill -l [exit_status]`

7300 *Obsolescent Versions:*

7301 `kill [-signal_name] pid ...`

7302 `kill [-signal_number] pid ...`

7303 **4.32.2 Description**

7304 The `kill` utility shall send a signal to the process(es) specified by each *pid*
7305 operand.

7306 For each *pid* operand, the `kill` utility shall perform actions equivalent to the
7307 POSIX.1 {8} `kill()` function called with the following arguments:

- 7308 (1) The value of the *pid* operand shall be used as the *pid* argument.
- 7309 (2) The *sig* argument is the value specified by the `-s` option, `-signal_number`
7310 option, or the `-signal_name` option, or by SIGTERM, if none of these
7311 options is specified.

7312 **4.32.3 Options**

7313 The `kill` utility shall conform to the utility argument syntax guidelines
7314 described in 2.10.2, except that in the obsolescent form, the `-signal_number` and
7315 `-signal_name` options are usually more than a single character.

7316 The following options shall be supported by the implementation:

- 7317 `-l` (The letter ell.) Write all values of *signal_name* supported by the
7318 implementation, if no operand is given. If an *exit_status* operand
7319 is given and it is a value of the `?` shell special parameter (see
7320 3.5.2 and `wait` in 4.70) corresponding to a process that was ter-
7321 minated by a signal, the *signal_name* corresponding to the signal
7322 that terminated the process shall be written. If an *exit_status*
7323 operand is given and it is the unsigned decimal integer value of a
7324 signal number, the *signal_name* (the POSIX.1 {8}-defined symbolic
7325 constant name without the SIG prefix) corresponding to that sig-
7326 nal shall be written. Otherwise, the results are unspecified.

7327 *-s signal_name*
 7328 Specify the signal to send, using one of the symbolic names
 7329 defined for Required Signals or Job Control Signals in POSIX.1 {8}
 7330 3.3.1.1. Values of *signal_name* shall be recognized in a case-
 7331 independent fashion, without the SIG prefix. In addition, the
 7332 symbolic name 0 shall be recognized, representing the signal
 7333 value zero. The corresponding signal shall be sent instead of
 7334 SIGTERM.

7335 *-signal_name*
 7336 (Obsolescent.) Equivalent to *-s signal_name*.

7337 *-signal_number*
 7338 (Obsolescent.) Specify a nonnegative decimal integer,
 7339 *signal_number*, representing the signal to be used instead of
 7340 SIGTERM, as the *sig* argument in the effective call to *kill()*. The
 7341 correspondence between integer values and the *sig* value used is
 7342 shown in the following table.

	<u><i>signal_number</i></u>	<u><i>sig</i> Value</u>
7343	0	0
7344	1	SIGHUP
7345	2	SIGINT
7346	3	SIGQUIT
7347	6	SIGABRT
7348	9	SIGKILL
7349	14	SIGALRM
7350	15	SIGTERM
7351		

7352 The effects of specifying any *signal_number* other than those
 7353 listed in the table are undefined.

7354 In the obsolescent versions, if the first argument is a negative integer, it shall be
 7355 interpreted as a *-signal_number* option, not as a negative *pid* operand specifying
 7356 a process group.

7357 **4.32.4 Operands**

7358 The following operands shall be supported by the implementation:

7359 *pid* A decimal integer specifying a process or process group to be sig-
 7360 naled. The process(es) selected by positive, negative, and zero
 7361 values of the *pid* operand shall be as described for POSIX.1 {8}
 7362 *kill()* function. If the first *pid* operand is negative, it should be
 7363 preceded by *--* to keep it from being interpreted as an option.

7364 *exit_status* A decimal integer specifying a signal number or the exit status of
7365 a process terminated by a signal.

7366 **4.32.5 External Influences**

7367 **4.32.5.1 Standard Input**

7368 None.

7369 **4.32.5.2 Input Files**

7370 None.

7371 **4.32.5.3 Environment Variables**

7372 The following environment variables shall affect the execution of `kill`:

7373 **LANG** This variable shall determine the locale to use for the
7374 locale categories when both **LC_ALL** and the correspond-
7375 ing environment variable (beginning with **LC_**) do not
7376 specify a locale. See 2.6.

7377 **LC_ALL** This variable shall determine the locale to be used to over-
7378 ride any values for locale categories specified by the set-
7379 tings of **LANG** or any environment variables beginning
7380 with **LC_**.

7381 **LC_CTYPE** This variable shall determine the locale for the interpreta-
7382 tion of sequences of bytes of text data as characters (e.g.,
7383 single- versus multibyte characters in arguments).

7384 **LC_MESSAGES** This variable shall determine the language in which mes-
7385 sages should be written.

7386 **4.32.5.4 Asynchronous Events**

7387 Default.

7388 **4.32.6 External Effects**

7389 **4.32.6.1 Standard Output**

7390 When the `-l` option is not specified, the standard output shall not be used.

7391 When the `-l` option is specified, the symbolic name of each signal shall be written
7392 in the following format:

7393 "%s%c", <signal_name>, <separator>

7394 where the <signal_name> is in uppercase, without the `SIG` prefix, and the

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

7395 <separator> shall be either a <newline> or a <space>. For the last signal writ-
 7396 ten, <separator> shall be a <newline>.

7397 When both the `-l` option and `exit_status` operand are specified, the symbolic name
 7398 of the corresponding signal shall be written in the following format:

7399 "%s\n", <signal_name>

7400 **4.32.6.2 Standard Error**

7401 Used only for diagnostic messages.

7402 **4.32.6.3 Output Files**

7403 None.

7404 **4.32.7 Extended Description**

7405 None.

7406 **4.32.8 Exit Status**

7407 The `kill` utility shall exit with one of the following values:

7408	0	At least one matching process was found for each <i>pid</i> operand, and the
7409		specified signal was successfully processed for at least one matching
7410		process.
7411	>0	An error occurred.

7412 **4.32.9 Consequences of Errors**

7413 Default.

7414 **4.32.10 Rationale.** *(This subclause is not a part of P1003.2)*

7415 **Examples, Usage**

7416 Any of the commands

```
7417     kill -9 100 -165
7418     kill -s kill 100 -165
7419     kill -s KILL 100 -165
```

7420 sends the SIGKILL signal to the process whose process ID is 100 and to all
 7421 processes whose process group ID is 165, assuming the sending process has per-
 7422 mission to send that signal to the specified processes, and that they exist.

7423 POSIX.1 {8} and POSIX.2 do not require specific signal numbers for any
 7424 *signal_names*. Even the `-signal_number` option provides symbolic (although

numeric) names for signals. If a process is terminated by a signal, its exit status indicates the signal that killed it, but the exact values are not specified. The `kill -l` option, however, can be used to map decimal signal numbers and exit status values into the name of a signal. The following example reports the status of a terminated job:

```

job
stat=$?
if [ $stat -eq 0 ]
then
    echo job completed successfully.
elif [ $stat -gt 128 ]
then
    echo job terminated by signal SIG$(kill -l $stat).
else
    echo job terminated with error code $stat.
fi

```

History of Decisions Made

The signal name extension was based on a desire to avoid limiting the `kill` utility to implementation-dependent values.

The `-l` option originated from the C-shell, and is also implemented in the KornShell. The C-shell output can consist of multiple output lines, because the signal names do not always fit on a single line on some terminal screens. The KornShell output also included the implementation-specific signal numbers, and was felt by the working group to be too difficult for scripts to parse conveniently. The specified output format is intended not only to accommodate the historical C-shell output, but also to permit an entirely vertical or entirely horizontal listing on systems for which this is appropriate.

An earlier draft invented the name `SIGNULL` as a *signal_name* for signal 0 (used by POSIX.1 {8} to test for the existence of a process without sending it a signal). Since the *signal_name* "0" can be used in this case unambiguously, `SIGNULL` has been removed.

An earlier draft also required symbolic *signal_names* to be recognized with or without the `SIG` prefix. Historical versions of `kill` have not written the `SIG` prefix for the `-l` option and have not recognized the `SIG` prefix on *signal_names*. Since neither application portability nor ease of use would be improved by requiring this extension, it is no longer required.

POSIX.2 contains no utility that browses for process IDs. Values for *pid* are available via the `!` and `$` parameters of the shell command language (see 3.5.2).

The use of numeric signal values was the subject of a long debate in the Working Group. During balloting, it was determined that their use should be declared obsolescent, but retained to provide backward compatibility to existing applications.

Existing implementations of `kill` permit negative *pid* operands representing process groups, but this was often unclearly documented. The assumption that an

initial negative number argument specifies a signal number (rather than a process group) is the existing behavior, and was retained. Therefore, to send the default signal to a process group (say 123), an application should use a command similar to one of the following:

```
kill -TERM -123
kill -- -123
```

The `-s` option was added in response to international interest in providing some form of `kill` that meets the Utility Syntax Guidelines.

Some implementations provide `kill` only as a shell built-in utility and use that status to support the extension of killing background asynchronous lists (those started with `&`), by the use of job identifiers. For example,

```
kill %1
```

would `kill` the first asynchronous list in the background. This standard does not require (but permits) such an extension, because other related job-control features are not provided by the shell, and because these facilities are not ordinarily usable in portable shell applications. This notation is expected to be introduced by the UPE.

4.33 `ln` — Link files

4.33.1 Synopsis

```
ln [-f] source_file target_file
```

```
ln [-f] source_file ... target_dir
```

4.33.2 Description

In the first synopsis form, the `ln` utility shall create a new directory entry (link) for the file specified by the *source_file* operand, at the *destination* path specified by the *target_file* operand. This first synopsis form shall be assumed when the final operand does not name an existing directory; if more than two operands are specified and the final is not an existing directory, an error shall result.

In the second synopsis form, the `ln` utility shall create a new directory entry for each file specified by a *source_file* operand, at a *destination* path in the existing directory named by *target_dir*.

If the last operand specifies an existing file of a type not specified by POSIX.1 {8}, the behavior is implementation defined.

The corresponding destination path for each *source_file* shall be the concatenation of the target directory pathname, a slash character, and the last pathname component of the *source_file*. The second synopsis form shall be assumed when the final operand names an existing directory.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

7505 For each *source_file*:

7506 (1) If the *destination* path exists:

7507 (a) If the `-f` option is not specified, `ln` shall write a diagnostic message
7508 to standard error, do nothing more with the current *source_file*, and
7509 go on to any remaining *source_files*.

7510 (b) Actions shall be performed equivalent to the POSIX.1 {8} `unlink()`
7511 function, called using *destination* as the *path* argument. If this fails
7512 for any reason, `ln` shall write a diagnostic message to standard
7513 error, do nothing more with the current *source_file*, and go on to any
7514 remaining *source_files*.

7515 (2) Actions shall be performed equivalent to the POSIX.1 {8} `link()` function
7516 using *source_file* as the *path1* argument, and the *destination* path as the
7517 *path2* argument.

7518 4.33.3 Options

7519 The `ln` utility shall conform to the utility argument syntax guidelines described
7520 in 2.10.2.

7521 The following option shall be supported by the implementation:

7522 `-f` Force existing *destination* pathnames to be removed to allow the
7523 link.

7524 4.33.4 Operands

7525 The following operands shall be supported by the implementation:

7526 *source_file* A pathname of a file to be linked. This can be a regular or special
7527 file; whether a directory can be linked is implementation defined.

7528 *target_file* The pathname of the new directory entry to be created.

7529 *target_dir* A pathname of an existing directory in which the new directory
7530 entries are to be created.

7531 4.33.5 External Influences

7532 4.33.5.1 Standard Input

7533 None.

4.33.5.2 Input Files

None.

4.33.5.3 Environment Variables

The following environment variables shall affect the execution of `ln`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

4.33.5.4 Asynchronous Events

Default.

4.33.6 External Effects**4.33.6.1 Standard Output**

None.

4.33.6.2 Standard Error

Used only for diagnostic messages.

4.33.6.3 Output Files

None.

4.33.7 Extended Description

None.

4.33.8 Exit Status

The `ln` utility shall exit with one of the following values:

- 0 All the specified files were linked successfully.
- >0 An error occurred.

4.33.9 Consequences of Errors

Default.

4.33.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

None.

History of Decisions Made

Some historic versions of `ln` (including the one specified by the *SVID*) unlink the destination file, if it exists, by default. If the mode does not permit writing, these versions will prompt for confirmation before attempting the unlink. In these versions the `-f` option causes `ln` to not attempt to prompt for confirmation.

This allows `ln` to succeed in creating links when the target file already exists, even if the file itself is not writable (although the directory must be). Previous versions of this draft specified this functionality.

This draft does not allow the `ln` utility to unlink existing destination paths by default for the following reasons:

- The `ln` utility has traditionally been used to provide locking for shell applications, a usage that is incompatible with `ln` unlinking the destination path by default. There was no corresponding technical advantage to adding this functionality.
- This functionality gave `ln` the ability to destroy the link structure of files, which changes the historical behavior of `ln`.
- This functionality is easily replicated with a combination of `rm` and `ln`.
- It is not historical practice in many systems; BSD and BSD-derived systems do not support this behavior. Unfortunately, whichever behavior is selected can cause scripts written expecting the other behavior to fail.
- It is preferable that `ln` perform in the same manner as the `link()` function, which does not permit the target to already exist.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

This standard retains the `-f` option to provide support for shell scripts depending on the *SVID* semantics. It seems likely that shell scripts would not be written to handle prompting by `ln`, and would therefore have specified the `-f` option.

It should also be noted that `-f` is an undocumented feature of many historical versions of the `ln` utility, allowing linking to directories. These versions will require modification.

Previous drafts of this standard also required an `-i` option, which behaved like the `-i` options in `cp` and `mv`, prompting for confirmation before unlinking existing files. This was not historical practice for the `ln` utility and has been deleted from this version.

Although symbolic links are not part of the standard, the `-s` option should be used only for the traditional purpose of creating symbolic links.

4.34 `locale` — Get locale-specific information

4.34.1 Synopsis

```
locale [-a | -m]
locale [-ck] name ...
```

4.34.2 Description

The `locale` utility shall write information about the current locale environment, or all public locales, to the standard output. For the purposes of this clause, a *public locale* is one provided by the implementation that is accessible to the application.

When `locale` is invoked without any arguments, it shall summarize the current locale environment for each locale category as determined by the settings of the environment variables defined in 2.5.

When invoked with operands, it shall write values that have been assigned to the keywords in the locale categories, as follows:

- Specifying a keyword name shall select the named keyword and the category containing that keyword.
- Specifying a category name shall select the named category and all keywords in that category.

4.34.3 Options

The `locale` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

- `-a` Write information about all available public locales. The available locales shall include `POSIX`, representing the POSIX Locale. The manner in which the implementation determines what other locales are available is implementation defined.
- `-c` Write the names of selected locale categories; see 4.34.6.1.
- `-k` Write the names and values of selected keywords. The implementation may omit values for some keywords; see 4.34.4.
- `-m` Write names of available charmaps; see 2.4.1.

1

4.34.4 Operands

The following operand shall be supported by the implementation:

- name* The name of a locale category as defined in 2.5, the name of a keyword in a locale category, or the reserved name `charmap`. The named category or keyword shall be selected for output. If a single *name* represents both a locale category name and a keyword name in the current locale, the results are unspecified. Otherwise, both category and keyword names can be specified as *name* operands, in any sequence. It is implementation defined whether any keyword values are written for the categories `LC_CTYPE` and `LC_COLLATE`.

4.34.5 External Influences

4.34.5.1 Standard Input

None.

4.34.5.2 Input Files

None.

4.34.5.3 Environment Variables

The following environment variables shall affect the execution of `locale`:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

7653 **LANG** This variable shall determine the locale to use for the
 7654 locale categories when both **LC_ALL** and the correspond-
 7655 ing environment variable (beginning with **LC_**) do not
 7656 specify a locale. See 2.6.

7657 **LC_ALL** This variable shall determine the locale to be used to over-
 7658 ride any values for locale categories specified by the set-
 7659 tings of **LANG** or any environment variables beginning
 7660 with **LC_**.

7661 **LC_CTYPE** This variable shall determine the locale for the interpreta-
 7662 tion of sequences of bytes of text data as characters (e.g.,
 7663 single- versus multibyte characters in arguments).

7664 **LC_MESSAGES** This variable shall determine the language in which mes-
 7665 sages should be written.

7666 The **LANG** and **LC_*** environment variables shall specify the current locale
 7667 environment to be written out; they shall be used if the **-a** option is not specified.

7668 **4.34.5.4 Asynchronous Events**

7669 Default.

7670 **4.34.6 External Effects**

7671 **4.34.6.1 Standard Output**

7672 If **locale** is invoked without any options or operands, the names and values of
 7673 the **LANG** and **LC_*** environment variables described in this standard shall be
 7674 written to the standard output, one variable per line, with **LANG** first, and each
 7675 line using the following format. Only those variables set in the environment and
 7676 not overridden by **LC_ALL** shall be written using this format:

7677 "**%s=%s\n**", *<variable_name>*, *<value>*

7678 The names of those **LC_*** variables associated with locale categories defined in
 7679 this standard that are not set in the environment or are overridden by **LC_ALL**
 7680 shall be written in the following format:

7681 "**%s=\" %s \"\n**", *<variable_name>*, *<implied value>*

7682 The *<implied value>* shall be the name of the locale that has been selected for
 7683 that category by the implementation, based on the values in **LANG** and **LC_ALL**,
 7684 as described in 2.6.

7685 The *<value>* and *<implied value>* shown above shall be properly quoted for possi- 1
 7686 ble later re-entry to the shell. The *<value>* shall not be quoted using double- 1
 7687 quotes (so that it can be distinguished by the user from the *<implied value>* case, 1
 7688 which always requires double-quotes). 1

7689 The **LC_ALL** variable shall be written last, using the first format shown above. If 1
 7690 it is not set, it shall be written as:

7691 "LC_ALL=\n"

7692 If any arguments are specified:

7693 (1) If the **-a** option is specified, the names of all the public locales shall be
 7694 written, each in the following format:

7695 "%s\n", *<locale name>*

7696 (2) If the **-c** option is specified, the name(s) of all selected categories shall be
 7697 written, each in the following format:

7698 "%s\n", *<category name>*

7699 If keywords are also selected for writing (see following items), the
 7700 category name output shall precede the keyword output for that category.

7701 If the **-c** option is not specified, the names of the categories shall not be 2
 7702 written; only the keywords, as selected by the *name* operand, shall be 2
 7703 written. 2

7704 (3) If the **-k** option is specified, the name(s) and value(s) of selected key-
 7705 words shall be written. If a value is nonnumeric, it shall be written in
 7706 the following format:

7707 "%s=\"%s\"\\n\", *<keyword name>*, *<keyword value>*

7708 If the keyword was charmap, the name of the charmap (if any) that was
 7709 specified via the `localedef -f` option when the locale was created shall
 7710 be written, with the word charmap as *<keyword name>*.

7711 If a value is numeric, it shall be written in one of the following formats:

7712 "%s=%d\\n\", *<keyword name>*, *<keyword value>*

7713 "%s=%c%o\\n\", *<keyword name>*, *<escape character>*,
 7714 *<keyword value>*

7715 "%s=%cx%x\\n\", *<keyword name>*, *<escape character>*,
 7716 *<keyword value>*

7717 where the *<escape character>* is that identified by the `escape_char` key-
 7718 word in the current locale; see 2.5.2.

7719 Compound keyword values (list entries) shall be separated in the output
 7720 by semicolons. When included in keyword values, the semicolon, the
 7721 double-quote, the backslash, and any control character shall be preceded
 7722 (escaped) with the escape character.

7723 (4) If the **-k** option is not specified, selected keyword values shall be written,
 7724 each in the following format:

7725 "%s\\n\", *<keyword value>*

7726 If the keyword was `charmap`, the name of the charmap (if any) that was
 7727 specified via the `localedef -f` option when the locale was created shall
 7728 be written.

7729 (5) If the `-m` option is specified, then a list of all available charmaps shall be
 7730 written, each in the format

7731 `"%s\n", <charmap>`

7732 where `<charmap>` is in a format suitable for use as the option-argument
 7733 to the `localedef -f` option.

7734 **4.34.6.2 Standard Error**

7735 Used only for diagnostic messages.

7736 **4.34.6.3 Output Files**

7737 None.

7738 **4.34.7 Extended Description**

7739 None.

7740 **4.34.8 Exit Status**

7741 The `locale` utility shall exit with one of the following values:

7742 0 All the requested information was found and output successfully.

7743 >0 An error occurred.

7744 **4.34.9 Consequences of Errors**

7745 Default.

7746 **4.34.10 Rationale.** *(This subclause is not a part of P1003.2)*

7747 **Examples, Usage**

7748 In the following examples, the assumption is that locale environment variables
 7749 are set as follows:

7750 `LANG=locale_x`

7751 `LC_COLLATE=locale_y`

7752 The command:

7753 `locale`

would result in the following output:

```
LANG=locale_x
LC_CTYPE="locale_x"
LC_COLLATE=locale_y
LC_TIME="locale_x"
LC_NUMERIC="locale_x"
LC_MONETARY="locale_x"
LC_MESSAGES="locale_x"
LC_ALL=
```

The order of presentation of the categories is not specified by this standard.

The command

```
LC_ALL=POSIX locale -ck decimal_point
```

would produce:

```
LC_NUMERIC
decimal_point="."
```

The following command shows an application of `locale` to determine whether a user supplied response is affirmative:

```
if printf "%s\n" "$response" | grep -Eq "$(locale yesexpr)"
then
    affirmative processing goes here
else
    nonaffirmative processing goes here
fi
```

If the **LANG** environment variable is not set or set to an empty value, or one of the **LC_*** environment variables is set to an unrecognized value, the actual locales assumed (if any) are implementation defined as described in 2.6.

Implementations are not required to write out the actual values for keywords in the categories **LC_CTYPE** and **LC_COLLATE**; however, they must write out the categories (allowing an application to determine, e.g., which character classes are available).

History of Decisions Made

This command was added in Draft 9 to resolve objections to the lack of a way for applications to determine what locales are available, a way to examine the contents of existing public locales, a way to retrieve specific locale items, and a way to recognize affirmative and negative responses in an international environment.

In Draft 10 it was cut back considerably in answer to balloting objections about its complexity and requirement of features not useful for application programs. The format for the no-arguments case was expanded to show the implied values of the categories as an aid to the novice user; the output was of little more value than that from `env`.

Based on the questionable value in a shell script of getting an entire array of characters back, and the problem of returning a collation description that makes sense, short of a complete `localedef` source, the output from requests for categories `LC_CTYPE` and `LC_COLLATE` has been made implementation defined.

The `-m` option has been added to allow applications to query for the existence of charmaps. The output is a list of the charmaps (implementation-supplied and user-supplied, if any) on the system.

The `-c` option was included for readability when more than one category is selected (e.g., via more than one keyword name or via a category name). It is valid both with and without the `-k` option.

The `charmap` keyword, which returns the name of the charmap (if any) that was used when the current locale was created, was introduced to allow applications needing the information to retrieve it.

4.35 `localedef` — Define locale environment

4.35.1 Synopsis

```
localedef [-c] [-f charmap] [-i sourcefile] name
```

4.35.2 Description

The `localedef` utility shall convert source definitions for locale categories into a format usable by the functions and utilities whose operational behavior is determined by the setting of the locale environment variables defined in 2.5. It is implementation defined whether users shall have the capability to create new locales, in addition to those supplied by the implementation. If the symbolic constant `{POSIX2_LOCALEDEF}` is defined, then the system supports the creation of new locales. In a system not supporting this capability, the `localedef` utility shall terminate with an exit code of 3.

The utility shall read source definitions for one or more locale categories belonging to the same locale from the file named in the `-i` option (if specified) or from standard input.

The *name* operand identifies the target locale. The utility shall support the creation of *public*, or generally accessible locales, as well as *private*, or restricted-access locales. Implementations may restrict the capability to create or modify public locales to users with the appropriate privileges.

Each category source definition shall be identified by the corresponding environment variable name and terminated by an `END category-name` statement. The following categories shall be supported. In addition, the input may contain source for implementation-defined categories.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

7830	LC_CTYPE	Defines character classification and case conversion.
7831	LC_COLLATE	Defines collation rules.
7832	LC_MONETARY	Defines the format and symbols used in formatting of monetary information.
7833		
7834	LC_NUMERIC	Defines the decimal delimiter, grouping, and grouping symbol for nonmonetary numeric editing.
7835		
7836	LC_TIME	Defines the format and content of date and time information.
7837		
7838	LC_MESSAGES	Defines the format and values of affirmative and negative responses.
7839		

7840 4.35.3 Options

7841 The `localedef` utility shall conform to the utility argument syntax guidelines
7842 described in 2.10.2.

7843 The following options shall be supported by the implementation:

7844 `-c` Create permanent output even if warning messages have been
7845 issued.

7846 `-f charmap` Specify the pathname of a file containing a mapping of character
7847 symbols and collating element symbols to actual character encod-
7848 ings. The format of the *charmap* is described under 2.4.1. This
7849 option shall be specified if symbolic names (other than collating
7850 symbols defined in a `collating-symbol` keyword) are used. If
7851 the `-f` option is not present, an implementation-defined default
7852 character mapping file shall be used.
7853

2

7854 `-i inputfile` The pathname of a file containing the source definitions. If this
7855 option is not present, source definitions shall be read from stan-
7856 dard input. The format of the *inputfile* is described in 2.5.2.

7857 4.35.4 Operands

7858 The following operand shall be supported by the implementation:

7859 *name* Identifies the locale. See 2.5 for a description of the use of this
7860 name. If the name contains one or more slash characters, *name*
7861 shall be interpreted as a pathname where the created locale
7862 definition(s) shall be stored. If *name* does not contain any slash
7863 characters, the interpretation of the name is implementation
7864 defined and the locale shall be public. This capability may be res-
7865 tricted to users with appropriate privileges.

4.35.5 External Influences

4.35.5.1 Standard Input

Unless the `-i` option is specified, the standard input shall be a text file containing one or more locale category source definitions, as described in 2.5.2. When lines are continued using the escape character mechanism, there is no limit to the length of the accumulated continued line.

4.35.5.2 Input Files

The character set mapping file specified as the *charmap* option-argument is described under 2.4.1. If a locale category source definition contains a `copy` statement, as defined in 2.5.2, and the `copy` statement names a valid, existing locale, then `localedef` shall behave as if the source definition had contained a valid category source definition for the named locale.

4.35.5.3 Environment Variables

The following environment variables shall affect the execution of `localedef`:

LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ . and LC_* variables as described in 2.6.
LC_COLLATE	(This variable shall have no affect on <code>localedef</code> ; the POSIX Locale shall be used for this category.)
LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of argument data as characters (e.g., single- versus multibyte characters). This variable shall have no affect on the processing of <code>localedef</code> input data; the POSIX Locale shall be used for this purpose, regardless of the value of this variable.
LC_MESSAGES	This variable shall determine the language in which messages should be written.

4.35.5.4 Asynchronous Events

Default.

7900 **4.35.6 External Effects**

7901 **4.35.6.1 Standard Output**

7902 The utility shall report all categories successfully processed, in an unspecified for-
7903 mat.

7904 **4.35.6.2 Standard Error**

7905 Used only for diagnostic messages.

7906 **4.35.6.3 Output Files**

7907 The format of the created output is unspecified. If the *name* operand does not
7908 contain a slash, the existence of an output file for the locale is unspecified.

7909 **4.35.7 Extended Description**

7910 None.

7911 **4.35.8 Exit Status**

7912 The `localedef` utility shall exit with one of the following values:

- | | | |
|------|----|--|
| 7913 | 0 | No errors occurred and the locale(s) were successfully created. |
| 7914 | 1 | Warnings occurred and the locale(s) were successfully created. |
| 7915 | 2 | The locale specification exceeded implementation limits or the coded |
| 7916 | | character set or sets used were not supported by the implementation, |
| 7917 | | and no locale was created. |
| 7918 | 3 | The capability to create new locales is not supported by the implementa- |
| 7919 | | tion. |
| 7920 | >3 | Warnings or errors occurred and no output was created. |

7921 **4.35.9 Consequences of Errors**

7922 If an error is detected, no permanent output shall be created.

7923 If warnings occur, permanent output shall be created if the `-c` option was
7924 specified. The following conditions shall cause warning messages to be issued:

- | | | |
|------|---|--|
| 7925 | — | If a symbolic name not found in the <i>charmap</i> file is used for the descrip- |
| 7926 | | tions of the LC_CTYPE or LC_COLLATE categories (for other categories, this |
| 7927 | | shall be an error conditions). |
| 7928 | — | If the number of operands to the <code>order</code> keyword exceeds the |
| 7929 | | { <code>COLL_WEIGHTS_MAX</code> } limit. |

7930 — If optional keywords not supported by the implementation are present in 1
 7931 the source. 1

7932 Other implementation-defined conditions may also cause warnings.

7933 **4.35.10 Rationale.** (*This subclause is not a part of P1003.2*)

7934 **Usage, Examples**

7935 The output produced by the `localedef` utility is implementation defined. The
 7936 *name* operand is used to identify the specific locale. (As a consequence, although
 7937 several categories can be processed in one execution, only categories belonging to
 7938 the same locale can be processed.)

7939 The *charmap* definition is optional, and is contained outside the locale definition.
 7940 This allows both completely “self-defined” source files, and “generic” sources
 7941 (applicable to more than one code set). To aid portability, all *charmap* definitions
 7942 shall use the same symbolic names for the portable character set. As explained in
 7943 2.4.1, it is implementation defined whether or not users or applications can pro-
 7944 vide additional character set description files. Therefore, the `-f` option might be
 7945 operable only when an implementation-provided *charmap* is named.

7946 **History of Decisions Made**

7947 This description is based on work performed in the UniForum Technical Commit-
 7948 tee Subcommittee on Internationalization.

7949 The `localedef` utility is provided as a standard, portable interface for imple-
 7950 mentations that allow users to create new locales, in addition to implementation-
 7951 supplied ones.

7952 The ability to create new locales and categories, already available on many com-
 7953 mercially available implementations of POSIX compliant systems, provides the
 7954 means by which application providers can develop portable applications which
 7955 use standard interfaces to adjust the behavior of the application to language and
 7956 culture differences.

4.36 `logger` — Log messages

4.36.1 Synopsis

`logger string...`

4.36.2 Description

The `logger` utility saves a message, in an unspecified manner and format, containing the *string* operands provided by the user. The messages are expected to be evaluated later by personnel performing system administration tasks.

4.36.3 Options

None.

4.36.4 Operands

The following operands shall be supported by the implementation:

<i>string</i>	One of the string arguments whose contents are concatenated together, in the order specified, separated by single <space>s.
---------------	---

4.36.5 External Influences

4.36.5.1 Standard Input

None.

4.36.5.2 Input Files

None.

4.36.5.3 Environment Variables

The following environment variables shall affect the execution of `logger`:

LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .

7985 **LC_CTYPE** This variable shall determine the locale for the interpreta-
 7986 tion of sequences of bytes of text data as characters (e.g.,
 7987 single- versus multibyte characters in arguments).

7988 **LC_MESSAGES** This variable shall determine the language in which diag-
 7989 nostic messages should be written.

7990 **4.36.5.4 Asynchronous Events**

7991 Default.

7992 **4.36.6 External Effects**

7993 **4.36.6.1 Standard Output**

7994 None.

7995 **4.36.6.2 Standard Error**

7996 Used only for diagnostic messages.

7997 **4.36.6.3 Output Files**

7998 Unspecified.

7999 **4.36.7 Extended Description**

8000 None.

8001 **4.36.8 Exit Status**

8002 The `logger` utility shall exit with one of the following values:

8003 0 Successful completion.

8004 >0 An error occurred.

8005 **4.36.9 Consequences of Errors**

8006 Default.

4.36.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

This utility allows logging of information for later use by a system administrator or programmer in determining why noninteractive utilities have failed. POSIX.2 makes no requirements for the locations of the saved message, their format, or retention period. It also provides no method for a portable application to read messages, once written. (It is expected that the POSIX.7 System Administration standard will have something to say about that.)

The purpose of this utility might best be illustrated by an example. A batch application, running noninteractively, tries to read a configuration file and fails; it may attempt to notify the system administrator with:

```
logger myname: unable to read file foo. [time stamp]
```

The text with **LC_MESSAGES** about diagnostic messages means diagnostics from logger to the user or application, not diagnostic messages that the user is sending to the system administrator.

History of Decisions Made

Multiple *string* arguments were allowed, similar to `echo`, for ease of use.

In Draft 9, the `posixlog` utility was renamed `logger` to match its BSD forebear, with which it is (downward) compatible.

The working group believed strongly that some method of alerting administrators to errors was necessary. The obvious example is a batch utility, running noninteractively, that is unable to read its configuration files, or that is unable to create or write its results file. However, the working group did not wish to define the format or delivery mechanisms as they have historically been (and will probably continue to be) very system specific, as well as involving functionality clearly outside of the scope of this standard.

Like the utilities `mailx` and `lp`, `logger` is admittedly difficult to test. This was not deemed sufficient justification to exclude these utilities from the standard. It is also arguable that they are, in fact, testable, but that the tests themselves are not portable.

8037 **4.37 logname — Return user's login name**

8038 **4.37.1 Synopsis**

8039 logname

8040 **4.37.2 Description**

8041 The logname utility shall write the user's login name to standard output. The
 8042 login name shall be the string that would be returned by the POSIX.1 {8} *getlo-*
 8043 *gin()* function. Under the conditions where the *getlogin()* function would fail, the
 8044 logname utility shall write a diagnostic message to standard error and exit with
 8045 a nonzero exit status.

8046 **4.37.3 Options**

8047 None.

8048 **4.37.4 Operands**

8049 None.

8050 **4.37.5 External Influences**

8051 **4.37.5.1 Standard Input**

8052 None.

8053 **4.37.5.2 Input Files**

8054 None.

8055 **4.37.5.3 Environment Variables**

8056 The following environment variables shall affect the execution of logname:

8057	LANG	This variable shall determine the locale to use for the
8058		locale categories when both LC_ALL and the correspond-
8059		ing environment variable (beginning with LC_) do not
8060		specify a locale. See 2.6.
8061	LC_ALL	This variable shall determine the locale to be used to over-
8062		ride any values for locale categories specified by the set-
8063		tings of LANG or any environment variables beginning
8064		with LC_ .

8065 **LC_MESSAGES** This variable shall determine the language in which mes-
8066 sages should be written.

8067 **4.37.5.4 Asynchronous Events**

8068 Default.

8069 **4.37.6 External Effects**

8070 **4.37.6.1 Standard Output**

8071 The `logname` utility output shall be a single line consisting of the user's login
8072 name:

8073 " %s\n", <login name>

8074 **4.37.6.2 Standard Error**

8075 Used only for diagnostic messages.

8076 **4.37.6.3 Output Files**

8077 None.

8078 **4.37.7 Extended Description**

8079 None.

8080 **4.37.8 Exit Status**

8081 The `logname` utility shall exit with one of the following values:

8082 0 Successful completion.

8083 >0 An error occurred.

8084 **4.37.9 Consequences of Errors**

8085 Default.

8086 **4.37.10 Rationale.** *(This subclause is not a part of P1003.2)*

8087 Examples, Usage

8088 The `logname` utility explicitly ignores the **LOGNAME** environment variable
8089 because environment changes could produce erroneous results.

8090 History of Decisions Made

8091 The `passwd` file is not listed as required, because the implementation may have
8092 other means of mapping login names.

8093 4.38 `lp` — Send files to a printer

8094 4.38.1 Synopsis

8095 `lp [-c] [-d dest] [-n copies] [file ...]`

8096 4.38.2 Description

8097 The `lp` utility shall copy the input files to an output device in an unspecified
8098 manner. The default output destination should be to a hardcopy device, such as a
8099 printer or microfilm recorder, that produces nonvolatile, human-readable docu-
8100 ments. If such a device is not available to the application, or if the system pro-
8101 vides no such device, the `lp` utility shall exit with a nonzero exit status.

8102 The actual writing to the output device may occur some time after the `lp` utility
8103 successfully exits. During the portion of the writing that corresponds to each
8104 input file, the implementation shall guarantee exclusive access to the device.

8105 4.38.3 Options

8106 The `lp` utility shall conform to the utility argument syntax guidelines described
8107 in 2.10.2.

8108 The following options shall be supported by the implementation:

8109	<code>-c</code>	Exit only after further access to any of the input files is no longer required. The application can then safely delete or modify the files without affecting the output operation.
8110		
8111		
8112	<code>-d <i>dest</i></code>	Specify a string that names the output device or destination. If
8113		<code>-d</code> is not specified, and neither the LPDEST nor PRINTER
8114		environment variable is set, an unspecified output device is used.
8115		The <code>-d <i>dest</i></code> option shall take precedence over LPDEST , which in
8116		turn shall take precedence over PRINTER . Results are undefined
8117		when <i>dest</i> contains a value that is not a valid device or

8118 destination name.

8119 **-n *copies*** Write *copies* number of copies of the files, where *copies* is a posi-
 8120 tive decimal integer. The methods for producing multiple copies
 8121 and for arranging the multiple copies when multiple *file* operands
 8122 are used are unspecified, except that each file shall be output as
 8123 an integral whole, not interleaved with portions of other files.

8124 **4.38.4 Operands**

8125 The following operands shall be supported by the implementation:

8126 ***file*** A pathname of a file to be output. If no *file* operands are
 8127 specified, or if a *file* operand is -, the standard input shall be
 8128 used. If a *file* operand is used, but the -c option is not specified,
 8129 the process performing the writing to the output device may have
 8130 user and group permissions that differ from that of the process
 8131 invoking **lp**.

8132 **4.38.5 External Influences**

8133 **4.38.5.1 Standard Input**

8134 The standard input shall be used only if no *file* operands are specified, or if a *file*
 8135 operand is -. See Input Files.

8136 **4.38.5.2 Input Files**

8137 The input files shall be text files.

8138 **4.38.5.3 Environment Variables**

8139 The following environment variables shall affect the execution of **lp**:

8140	LANG	This variable shall determine the locale to use for the
8141		locale categories when both LC_ALL and the correspond-
8142		ing environment variable (beginning with LC_) do not
8143		specify a locale. See 2.6.
8144	LC_ALL	This variable shall determine the locale to be used to over-
8145		ride any values for locale categories specified by the set-
8146		tings of LANG or any environment variables beginning
8147		with LC_ .
8148	LC_CTYPE	This variable shall determine the locale for the interpreta-
8149		tion of sequences of bytes of text data as characters (e.g.,
8150		single- versus multibyte characters in arguments and
8151		input files).

8152	LC_MESSAGES	This variable shall determine the language in which mes-
8153		sages should be written.
8154	LPDEST	This variable shall be interpreted as a string that names
8155		the output device or destination. If the LPDEST environ-
8156		ment variable is not set, the PRINTER environment vari-
8157		able shall be used. The <code>-d <i>dest</i></code> option shall take pre-
8158		cedence over LPDEST . Results are undefined when <code>-d</code> is
8159		not specified and LPDEST contains a value that is not a
8160		valid device or destination name.
8161	PRINTER	This variable shall be interpreted as a string that names
8162		the output device or destination. If the LPDEST and
8163		PRINTER environment variables are not set, an
8164		unspecified output device is used. The <code>-d <i>dest</i></code> option and
8165		the LPDEST environment variable shall take precedence
8166		over PRINTER . Results are undefined when <code>-d</code> is not
8167		specified, LPDEST is unset, and PRINTER contains a
8168		value that is not a valid device or destination name.

8169 **4.38.5.4 Asynchronous Events**

8170 Default.

8171 **4.38.6 External Effects**

8172 **4.38.6.1 Standard Output**

8173 A message concerning the identification or status of the print request may be 2
8174 written, in an unspecified format. 2

8175 **4.38.6.2 Standard Error**

8176 Used only for diagnostic messages.

8177 **4.38.6.3 Output Files**

8178 None.

8179 **4.38.7 Extended Description**

8180 None.

4.38.8 Exit Status

The `lp` utility shall exit with one of the following values:

- 0 All input files were processed successfully.
- >0 No output device was available, or an error occurred.

4.38.9 Consequences of Errors

Default.

4.38.10 Rationale. *(This subclause is not a part of P1003.2)***Examples, Usage**

Since the default destination, device type, queueing mechanisms, and acceptable forms of input are all unspecified, usage guidelines for what a portable application can do are as follows:

- (1) Use the command in a pipeline, or with `-c`, so that there are no permission problems and the files can be safely deleted or modified.
- (2) Limit output to text files of reasonable line lengths and printable characters and include no device-specific formatting information, such as a page description language. The meaning of “reasonable” in this context can only be answered as a quality of implementation issue, but should be apparent from historical usage patterns in the industry and the locale. The `pr` and `fold` utilities can be used to achieve reasonable formatting for the implementation’s default page size.

Alternatively, the application can arrange its installation in such a way that requires the system administrator or operator to provide the appropriate information on `lp` options and environment variable values.

At a minimum, having this utility in the standard tells the industry that portable applications require a means to print output and provides at least a command name and **LPDEST** routing mechanism that can be used for discussions between vendors, application writers, and users. The use of “should” in the Description clearly shows the working group’s intent, even if it cannot mandate that all systems (such as laptops) have printers.

Examples:

To print file *file*:

```
lp -c file
```

To print multiple files with headers:

```
pr file1 file2 | lp
```

On most existing implementations of `lp`, an option is provided to pass printer specific options to the daemon handling the printer. It is not specified here

because the printer-specific options are widespread and in conflict, the `lp` specified here is not required to even have a queueing mechanism, and the choice of options varies widely from printer to printer. Nonetheless, implementors are encouraged to use this mechanism where appropriate:

`-o option` Specifies an implementation-defined option that controls the specific operation of the printer. The following *options* could be used for the meanings below if the hardware is capable of supporting the option.

<i>option</i>	Meaning
<code>lp2</code>	two logical pages per physical page
<code>lp4</code>	four logical pages per physical page
<code>d</code>	double sided

POSIX.2 does not specify what the ownership of the process performing the writing to the output device may be. If `-c` is not used, it is unspecified whether the process performing the writing to the output device will have permission to read *file* if there are any restrictions in place on who may read *file* until after it is printed. Also, if `-c` is not used, the results of deleting *file* before it is printed are unspecified.

History of Decisions Made

The `lp` utility was designed to be a basic version of a utility that is already available in many historical implementations. The working group felt that it should be implementable simply as:

```
cat "$@" > /dev/lp
```

after appropriate processing of options, if that is how the implementation chose to do it and if exclusive access could be granted (so that two users did not write to the device simultaneously). Although in the future the working group may add other options to this utility, it should always be able to execute with no options or operands and send the standard input to an unspecified output device.

The standard makes no representations concerning the format of the printed output, except that it must be “human-readable” and “nonvolatile.” Thus, writing by default to a disk or tape drive or a display terminal would not qualify. (Such destinations are not prohibited when `-d dest`, **LPDEST**, or **PRINTER** are used, however.)

A portable application will use one of the *file* operands only with the `-c` option or if the file is publicly readable and guaranteed to be available at the time of printing. This is because the standard gives the implementation the freedom to queue up the request for printing at some later time by a different process that might not be able to access the file.

The standard is worded such that a “print job” consisting of multiple input files, possibly in multiple copies, is guaranteed to print so that any one file is not jumbled up with another, but there is no statement that all the files or copies have to

8258 print out together.

8259 The `-c` option may imply a spooling operation, but this is not required. The util-
8260 ity can be implemented to simply wait until the printer is ready and then wait
8261 until it's finished. Because of that, there is no attempt to define a queueing
8262 mechanism (priorities, classes of output, etc.).

8263 The `-n` and `-d` options were added in response to balloting objections that too lit-
8264 tle historical value was being provided.

8265 Although the historical System V `lp` and BSD `lpr` utilities have provided similar
8266 functionality, they used different names for the environment variable specifying
8267 the destination printer. Since the name of the utility here is `lp`, **LPDEST** (used
8268 by the System V `lp` utility) was given precedence over **PRINTER** (used by the BSD
8269 `lpr` utility). Since environments of users frequently contain one or the other
8270 environment variable, the `lp` utility is required to recognize both. If this was not
8271 done, many applications would send output to unexpected output devices when
8272 users moved from system to system.

8273 Some have commented that `lp` has far too little functionality to make it
8274 worthwhile. Requests have proposed additional options or operands or both that
8275 added functionality. The requests included:

- 8276 — wording *requiring* the output to be “hardcopy”
- 8277 — a requirement for multiple printers
- 8278 — options for PostScript, `dimpres`, `hp`, and lineprint formats

8279 Given that a POSIX.2 compliant system is not required to even have a printer,
8280 placing further restrictions upon the behavior of the printer is not useful. Since
8281 hardcopy format is so application dependent, it is difficult, if not impossible, to
8282 select a reasonable subset of functionality that should be required on all POSIX.2
8283 compliant systems.

8284 The term “unspecified” is used in this clause in lieu of “implementation defined”
8285 as most known implementations would not be able to say anything fully useful in
8286 their conformance documents: the existence and usage of printers is very depen-
8287 dent on how the system administrator configures each individual system.

8288 **4.39 `ls` — List directory contents**

8289 **4.39.1 Synopsis**

8290 `ls [-CFRacdilqrtul] [file ...]`

8291 **4.39.2 Description**

8292 For each operand that names a file of a type other than directory, `ls` shall write
 8293 the name of the file as well as any requested, associated information. For each
 8294 operand that names a file of type directory, `ls` shall write the names of files con-
 8295 tained within that directory, as well as any requested, associated information.

8296 If no operands are specified, the contents of the current directory shall be written.
 8297 If more than one operand is specified, nondirectory operands shall be written first;
 8298 directory and nondirectory operands shall be sorted separately according to the
 8299 collating sequence in the current locale.

8300 **4.39.3 Options**

8301 The `ls` utility shall conform to the utility argument syntax guidelines described
 8302 in 2.10.2.

8303 The following options shall be supported by the implementation:

8304	<code>-C</code>	Write multi-text-column output with entries sorted down the columns, according to the collating sequence. The number of text columns and the column separator characters are unspecified, but should be adapted to the nature of the output device.	
8305			
8306			
8307			
8308	<code>-F</code>	Write a slash (/) immediately after each pathname that is a directory, an asterisk (*) after each that is executable, and a vertical bar () after each that is a FIFO.	
8309			
8310			
8311	<code>-R</code>	Recursively list subdirectories encountered.	
8312	<code>-a</code>	Write out all directory entries, including those whose names begin with a period (.). Entries beginning with a period (.) shall not be written out unless explicitly referenced, the <code>-a</code> option is supplied, or an implementation-defined condition causes them to be written.	
8313			
8314			
8315			
8316			
8317	<code>-t</code>	Use time of last modification of the file status information (see POSIX.1 {8} 5.6.1.3) instead of last modification of the file itself for sorting (<code>-t</code>) or writing (<code>-l</code>).	
8318			
8319			
8320	<code>-d</code>	Do not treat directories differently than other types of files. The use of <code>-d</code> with <code>-R</code> produces unspecified results.	2
8321			2

8322	<code>-i</code>	For each file, write the file's file serial number (see POSIX.1 {8} 5.6.2).	
8323			
8324	<code>-l</code>	(The letter ell.) Write out in long format (see 4.39.6.1). When <code>-l</code>	2
8325		(ell) is specified, <code>-l</code> (one) shall be assumed.	2
8326	<code>-q</code>	Force each instance of nonprintable filename characters and	2
8327		<code><tab>s</code> to be written as the question-mark (?) character. Imple-	
8328		mentations may provide this option by default if the output is to a	
8329		terminal device.	
8330	<code>-r</code>	Reverse the order of the sort to get reverse collating sequence or	
8331		oldest first.	
8332	<code>-t</code>	Sort by time modified (most recently modified first) before sorting	
8333		the operands by the collating sequence.	
8334	<code>-u</code>	Use time of last access (see POSIX.1 {8} 5.6.1.3) instead of last	
8335		modification of the file for sorting (<code>-t</code>) or writing (<code>-l</code>).	
8336	<code>-1</code>	(The numeric digit one.) Force output to be one entry per line.	
8337	Specifying more than one of the options in the following mutually exclusive pairs		2
8338	shall not be considered an error: <code>-C</code> and <code>-l</code> (ell), <code>-C</code> and <code>-1</code> (one), <code>-c</code> and <code>-u</code> . The		2
8339	last option specified in each pair shall determine the output format.		2

8340 4.39.4 Operands

8341 The following operands shall be supported by the implementation:

8342	<i>file</i>	A pathname of a file to be written. If the file specified is not
8343		found, a diagnostic message shall be output on standard error.

8344 4.39.5 External Influences

8345 4.39.5.1 Standard Input

8346 None.

8347 4.39.5.2 Input Files

8348 None.

8349 4.39.5.3 Environment Variables

8350 The following environment variables shall affect the execution of `ls`:

8351	COLUMNS	This variable shall determine the user's preferred column
8352		position width for writing multiple-text-column output. If
8353		this variable contains a string representing a decimal
8354		integer, the <code>ls</code> utility shall calculate how many pathname

8355		text columns to write (see <code>-C</code>) based on the width provided.
8356		If COLUMNS is not set or invalid, an implementation-defined number of column positions shall
8357		be assumed, based on the implementation's knowledge of
8358		the output device. The column width chosen to write the
8359		names of files in any given directory shall be constant.
8360		File names shall not be truncated to fit into the multiple-
8361		text-column output.
8362		
8363	LANG	This variable shall determine the locale to use for the
8364		locale categories when both LC_ALL and the correspond-
8365		ing environment variable (beginning with LC_) do not
8366		specify a locale. See 2.6.
8367	LC_ALL	This variable shall determine the locale to be used to over-
8368		ride any values for locale categories specified by the set-
8369		tings of LANG or any environment variables beginning
8370		with LC_ .
8371	LC_COLLATE	This variable shall determine the locale for character col-
8372		lation information in determining the pathname collation
8373		sequence.
8374	LC_CTYPE	This variable shall determine the locale for the interpreta-
8375		tion of sequences of bytes of text data as characters (e.g.,
8376		single- versus multibyte characters in arguments) and
8377		which characters are defined as printable (character class
8378		<code>print</code>).
8379	LC_MESSAGES	This variable shall determine the language in which mes-
8380		sages should be written.
8381	LC_TIME	This variable shall determine the the format and contents
8382		for date and time strings written by <code>ls</code> .
8383	TZ	This variable shall determine the time zone for date and
8384		time strings written by <code>ls</code> .

8385 **4.39.5.4 Asynchronous Events**

8386 Default.

8387 **4.39.6 External Effects**

8388 **4.39.6.1 Standard Output**

8389 The default format shall be to list one entry per line to standard output; the
 8390 exceptions are to terminals or when the `-C` option is specified. If the output is to a
 8391 terminal, the format is implementation defined.

2

8392 If the `-i` option is specified, the file's file serial number (see POSIX.1 {8} 5.6.1)
 8393 shall be written in the following format before any other output for the
 8394 corresponding entry:

2

8395 "`%u` ", *<file serial number>*

2

8396 If the `-l` option is specified, the following information shall be written:

8397 "`%s %u %s %s %u %s %s\n`", *<file mode>*, *<number of links>*,
 8398 *<owner name>*, *<group name>*, *<number of bytes in the file>*,
 8399 *<date and time>*, *<pathname>*

1

8400 If *<owner name>* or *<group name>* cannot be determined, they shall be replaced
 8401 with their associated numeric values using the format "`%u`".

8402 The *<date and time>*, field shall contain the appropriate date and time stamp of
 8403 when the file was last modified. In the POSIX Locale, the field shall be the
 8404 equivalent of the output of the following `date` command (see 4.15):

8405 `date "+%b %e %H:%M"`

8406 if the file has been modified in the last six months, or:

8407 `date "+%b %e %Y"`

8408 (where two `<space>` characters are used between `%e` and `%Y`) if the file has not
 8409 been modified in the last six months or if the modification date is in the future,
 8410 except that, in both cases, the final `<newline>` produced by `date` shall not be
 8411 included and the output shall be as if the `date` command were executed at the
 8412 time of the last modification date of the file rather than the current time. When
 8413 the `LC_TIME` locale category is not set to the POSIX Locale, a different format and
 8414 order of presentation of this field may be used.

8415 If the file is a character special or block special file, the size of the file may be
 8416 replaced with implementation-defined information associated with the device in
 8417 question.

8418 If the *pathname* was specified as a *file* operand, it shall be written as specified.

8419 The file mode written under the `-l` option shall consist of the following format:

8420 "`%c%s%s%s%s`", *<entry type>*, *<owner permissions>*,
 8421 *<group permissions>*, *<other permissions>*,
 8422 *<optional alternate access method flag>*

8423 The *<optional alternate access method flag>* shall be a single `<space>` if there is
 8424 no alternate or additional access control method associated with the file; other-
 8425 wise, a printable character shall be used.

8426 The *<entry type>* character shall describe the type of file, as follows:

8427	d	Directory
8428	b	Block special file
8429	c	Character special file
8430	p	FIFO
8431	–	Regular file

8432 Implementations may add other characters to this list to represent other,
8433 implementation-defined, file types.

8434 The next three fields shall be three characters each:

8435	<i><owner permissions></i>	Permissions for the file owner class (see 2.9.1.3).
8436	<i><group permissions></i>	Permissions for the file group class.
8437	<i><other permissions></i>	Permissions for the file other class.

8438 Each field shall have three character positions:

- | | | |
|------|-----|---|
| 8439 | (1) | If <i>r</i> , the file is readable; if –, it is not readable. |
| 8440 | (2) | If <i>w</i> , the file is writable; if –, it is not writable. |
| 8441 | (3) | The first of the following that applies: |
| 8442 | S | If in <i><owner permissions></i> , the file is not executable and set-user-ID mode is set. If in <i><group permissions></i> , the file is not |
| 8443 | | executable and set-group-ID mode is set. |
| 8444 | | |
| 8445 | s | If in <i><owner permissions></i> , the file is executable and set-user-ID |
| 8446 | | mode is set. If in <i><group permissions></i> , the file is executable |
| 8447 | | and set-group-ID mode is set. |
| 8448 | x | The file is executable or the directory is searchable. |
| 8449 | – | None of the attributes of S, s, or x applies. |

8450 Implementations may add other characters to this list for the third char-
8451 acter position. Such additions shall, however, be written in lowercase if
8452 the file is executable or searchable, and in uppercase if it is not.

8453 If the *–l* option is specified, each list of files within the directory shall be preceded
8454 by a status line indicating the number of file system blocks occupied by files in the
8455 directory in 512-byte units, rounded up to the next integral number of units, if
8456 necessary. In the POSIX Locale, the format shall be:

8457 "total %u\n", *<number of units in the directory>*

8458 If more than one directory, or a combination of nondirectory files and directories
8459 are written, either as a result of specifying multiple operands, or the *–R* option,
8460 each list of files within a directory shall be preceded by:

8461 "\n%s:\n", *<directory name>*

8462 If this string is the first thing to be written, the first <newline> character shall
8463 not be written. This output shall precede the number of units in the directory.

8464 **4.39.6.2 Standard Error**

8465 Used only for diagnostic messages.

8466 **4.39.6.3 Output Files**

8467 None.

8468 **4.39.7 Extended Description**

8469 None.

8470 **4.39.8 Exit Status**

8471 The `ls` utility shall exit with one of the following values:

8472 0 All files were written successfully.

8473 >0 An error occurred.

8474 **4.39.9 Consequences of Errors**

8475 Default.

8476 **4.39.10 Rationale.** *(This subclause is not a part of P1003.2)*

8477 **Examples, Usage**

8478 An example of a small directory tree being fully listed with `ls -laRF a` in the
8479 POSIX Locale:

```
8480      total 11
8481      drwxr-xr-x  3 hlj      prog          64 Jul  4 12:07 ./
8482      drwxrwxrwx  4 hlj      prog        3264 Jul  4 12:09 ../
8483      drwxr-xr-x  2 hlj      prog          48 Jul  4 12:07 b/
8484      -rwxr--r--  1 hlj      prog        572 Jul  4 12:07 foo*

8485      a/b:
8486      total 4
8487      drwxr-xr-x  2 hlj      prog          48 Jul  4 12:07 ./
8488      drwxr-xr-x  3 hlj      prog          64 Jul  4 12:07 ../
8489      -rw-r--r--  1 hlj      prog       700 Jul  4 12:07 bar
```

8490 Many implementations use the equals-sign (=) and the at-sign (@) to denote sock-
8491 ets bound to the file system and symbolic links, respectively, for the `-F` option.
8492 Similarly, many historical implementations use the “s” character and the “l”

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

character to denote sockets and symbolic links, respectively, as the entry type characters for the `-l` option. These characters should not be used to signify any other types of files in new implementations.

It is difficult for an application to use every part of the file modes field of `ls -l` in a portable manner. Certain file types and executable bits are not guaranteed to be exactly as shown, as implementations may have extensions. Applications can use this field to pass directly to a user printout or prompt, but actions based on its contents should generally be deferred, instead, to the `test` utility (see 4.62).

The output of `ls` (with the `-l` option) contains information that logically could be used by utilities such as `chmod` and `touch` to restore files to a known state. However, this information is presented in a format that cannot be used directly by those utilities or be easily translated into a format that can be used. In POSIX.2, a character was added to the end of the permissions string so that applications will at least have an indication that they may be working in an area they do not understand instead of assuming that they can translate the permissions string into something that can be used. POSIX.6 may define one or more specific characters to be used based on different standard additional or alternative access control mechanisms.

Some historical implementations of the `ls` utility show all entries in a directory except dot and dot-dot when super-user invokes `ls` without specifying the `-a` option. When “normal” users invoke `ls` without specifying `-a`, they should not see information about any files with names beginning with period unless they were named as file operands.

As with many of the utilities that deal with file names, the output of `ls` for multiple files or in one of the long listing formats must be used carefully on systems where file names can contain embedded white space. It is recommended that systems and system administrators institute policies and user training to limit the use of such file names.

History of Decisions Made

Implementations are expected to traverse arbitrary depths when processing the `-R` option. The only limitation on depth should be based on running out of physical storage for keeping track of untraversed directories.

The `-1` (one) option is currently found in BSD and BSD-derived implementations only. It was required in the standard so that portable applications might ensure that output is one entry per line, even if the output is to a terminal. Recent changes to the 2.10.2 allow numeric options.

Generally, the standard is mute about what happens when options are given multiple times. In the case of `-C`, `-l`, and `-1`, however, it does specify the results of these overlapping options. Since `ls` is one of the most aliased commands, it is important that the implementation do the correct thing. For example, if the alias were

```
alias ls="ls -C"
```

and the user typed “`ls -1`”, single text column output should result, not an error.

(The working group is aware that aliases are not included in the standard; this is just an example.)

The *SVID* defines a `-x` option for multi-text-column output sorted horizontally. The working group felt that `-x` provided only limited increased functionality over the `-C` option. The *SVID* also provides a `-m` option for a comma separated list of files. It was not provided because similar functionality (easier to parse for scripts) can be provided by the `echo` and `printf` utilities. Nonetheless, implementations considering adding new options to `ls` should look at historical BSD and System V versions of `ls` to avoid naming conflicts.

The BSD `ls` provides a `-A` option (like `-a`, but dot and dot-dot are not written out). The small difference from `-a` did not seem important enough to require both.

Implementations are allowed to make `-q` the default for terminals to prevent Trojan Horse attacks on terminals with special escape sequences. This is not required because:

- Some control characters may be useful on some terminals; for example, a system might write them as `\001` or `^A`,
- Special behavior for terminals is not relevant to application portability.

The `-s` option provided by existing implementations is not required by this standard. The number of disk blocks occupied by the file that it reports varies depending on underlying file system type, block size units reported, and the method of calculating the number of blocks. On some file system types, the number is the actual number of blocks occupied by the file (counting indirect blocks and ignoring holes in the file); on others it is calculated based on the file size (usually making an allowance for indirect blocks, but ignoring holes). The former is probably more useful, but depends on information not required by POSIX.1 {8} and not readily accessible on some file system types. Therefore, applications cannot depend on `-s` to provide any portable information. Implementations are urged to continue to provide this option, but applications should use the file size reported by the `-l` option in any calculations about the space needed to store a file.

An earlier draft specified that the optional alternate access method flag had to be `“+”` if there was an alternate access method used on the file or `<space>` if there was not. This was changed in Draft 10 to be `<space>` if there is not and a single printable character if there is. This was done for three reasons: 1) There are existing implementations using characters other than `“+”`; 2) There are implementations that vary this character used in that position to distinguish between various alternate access methods in use, and; 3) the developers of the standard did not want to preclude specification by POSIX.6 that might need a way to specify more than one alternate access method. Nonetheless, implementations providing a single alternate access method are encouraged to use `“+”`.

In a previous draft the units used to specify the number of blocks occupied by files in a directory in an `ls -l` listing was implementation defined. This was because BSD systems have historically used 1024-byte units and System V systems have historically used 512-byte units. It was pointed out by developers at Berkeley

that BSD has used 512-byte units in some places and 1024-byte units in other places. (System V has consistently used 512.) Therefore, POSIX.2 and POSIX.2a usually specify 512 and that value has been restored here as it was in Draft 9. Future releases of BSD are expected to consistently provide 512 as a default with a way of specifying 1024-byte units where appropriate.

The *<date and time>* field in the *-l* format is specified only for the POSIX Locale. As noted, the format can be different in other locales. No mechanism for defining this is present in this standard, as the appropriate vehicle is a messaging system; i.e., the format should be specified as a “message.”

4.40 mailx — Process messages

4.40.1 Synopsis

`mailx [-s subject] address ...`

4.40.2 Description

The `mailx` utility shall read standard input and send it to one or more addresses in an unspecified manner. Unless the first character of one or more lines is tilde (~), all characters in the input message shall appear in the delivered message, but additional characters may be inserted in the message before it is retrieved.

4.40.3 Options

The `mailx` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following option shall be supported by the implementation:

<code>-s <i>subject</i></code>	A string representing the subject of the message. All characters	2
	in the <i>subject</i> string shall appear in the delivered message. The	2
	results are unspecified if <i>subject</i> is longer than {LINE_MAX} – 10	2
	bytes or contains a <i><newline></i> .	2

4.40.4 Operands

The following operand shall be supported by the implementation:

<i>address</i>	Send a message to <i>address</i> . Valid login names on the local system shall be accepted as valid <i>addresses</i> . The interpretation of other types of <i>addresses</i> is unspecified. An implementation-defined way for a user with a login-name address to retrieve the message shall be provided by the implementation.
----------------	--

4.40.5 External Influences

4.40.5.1 Standard Input

The standard input shall be a text file. The results are unspecified if the first character of any input line is a tilde (~).

4.40.5.2 Input Files

None.

4.40.5.3 Environment Variables

The following environment variables shall affect the execution of `mailx`:

DEAD	This variable shall affect the processing of signals by <code>mailx</code> : if the application sets this variable to <code>/dev/null</code> , the results of receiving a signal are as described by this standard; they are otherwise unspecified.
HOME	This variable shall be interpreted as a pathname of the user's home directory.
LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).
LC_MESSAGES	This variable shall determine the language in which messages should be written.
MAILRC	This variable shall affect the startup processing of <code>mailx</code> : if the application sets this variable to <code>/dev/null</code> , <code>mailx</code> shall operate as described by this standard; otherwise, unspecified results occur.

4.40.5.4 Asynchronous Events

Default.

8646 **4.40.6 External Effects**

8647 **4.40.6.1 Standard Output**

8648 None.

8649 **4.40.6.2 Standard Error**

8650 Used only for diagnostic messages.

8651 **4.40.6.3 Output Files**

8652 None.

8653 **4.40.7 Extended Description**

8654 None.

8655 **4.40.8 Exit Status**

8656 The `mailx` utility shall exit with one of the following values:

8657 0 Successful completion.

8658 >0 An error occurred.

8659 **4.40.9 Consequences of Errors**

8660 Default.

8661 **4.40.10 Rationale.** *(This subclause is not a part of P1003.2)*

8662 **Usage, Examples**

8663 The intent is that a header indicating who sent the message and a message sub-
 8664 ject string, the contents of the standard input, and perhaps a trailer is delivered
 8665 to users specified by the given addresses. The standard input, however, may have
 8666 to be manipulated slightly to avoid confusion between message text and headers
 8667 as it passes through the message delivery system. POSIX.2 does not specify how
 8668 standard input may be manipulated; that will be specified in detail by POSIX.2a.

8669 The restriction on a subject line being {LINE_MAX} – 10 bytes is based on the his- 2
 8670 torical format that consumes 10 bytes for "Subject: " and the trailing <new- 2
 8671 line>. Many historical mailers that a message may encounter on other systems 2
 8672 will not be able to handle lines that long, however. 2

History of Decisions Made

The developers of the standard felt strongly that a method for applications to send messages to specific users was necessary. The obvious example is a batch utility, running noninteractively, that wishes to communicate errors or results to a user. However, the actual format, delivery mechanism, and method of reading the message are clearly beyond the scope of this standard.

The intent of this command is to provide a simple, portable interface for sending messages noninteractively. It merely defines a “front-end” to the historical mail system. It is suggested that implementations explicitly denote the sender and recipient in the body of the delivered message. Further specification of formats for either the message envelope or the message itself were deliberately not made, as the industry is in the midst of changing from the current standards to a more internationalized standard and it is probably incorrect, at this time, to require either one.

Implementations are encouraged to conform to the various delivery mechanisms described in ARPANET Requests for Comment Numbers 819, 822, 882, 920, 921, and the CCITT X.400 standards.

The standard does not place any restrictions on the length of messages handled by `mailx`, and for delivery of local messages the only limitations should be the normal problems of available disk space for the target mail file. When sending messages to external machines, applications are advised to limit messages to less than 50 kilobytes because many mail gateways impose message-length restrictions. (Note that this is usually an administrative issue based on the amount of mail traffic and disk space available on the gateways. Therefore, there is no way for this standard to require implementations to guarantee delivery of long messages to remote systems.)

Like the utilities `logger` and `lp`, `mailx` is admittedly difficult to test. This was not deemed sufficient justification to exclude these utilities from the standard. It is also arguable that they are, in fact, testable, but that the tests themselves are not portable.

Before Draft 7, there was a utility named `mailto`. In Draft 7, the name was changed to `sendto` because of comments noting that `mailto` implied full mail-like functionality and that was not what the specification provided. However, there have been consistent comments that it does not make sense to end up with a standard that will require two mail-sending interfaces. (POSIX.2a is working on a fully fleshed-out mail-sending and -reading utility based on the historical System V `mailx` utility.) A message- (or mail-) sending utility that is a subset of the interactive utility that will be described by POSIX.2a is much more consistent with the rest of the standard. Therefore, in Draft 10 the name has been changed again to `mailx` and the description is a small subset of the functionality being specified by POSIX.2a. It provides a portable way for a shell script to be able to send a message to a user on the local system. It is expected that implementations that have provided `mailx` in the past will use it to meet the POSIX.2 requirements. Implementations that have not provided `mailx` in the past will be able to create a simple interface to their current mailer to meet these requirements.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

Most of the features provided by `mailx` (and the similar BSD `Mail`) utility are not specified here because they are not needed for noninteractive use (applications do not usually read mail without user participation) and they depend on other interactive features that are not defined by POSIX.2, but will be defined by POSIX.2a (the `~v` command, for instance, uses the `vi` editor as a default.)

If the **DEAD** environment variable is not set to `/dev/null`, historical versions of `mailx` and `Mail` save a message being constructed in a file under some circumstances when some asynchronous events occur. The details will be specified by POSIX.2a.

If the **MAILRC** environment variable does not name an empty file, historical versions of `mailx` and `Mail` read initialization commands from a file before processing begins. Since the initialization that a user specifies could alter the contents of messages an application is trying to send, applications are advised to set **MAILRC** to `/dev/null`. POSIX.2a will specify details on the format of the initialization file.

Options to specify addresses as “cc” (carbon-copy) or “bcc” (blind-carbon-copy) were considered to be format details and were omitted.

A zero exit status implies that all messages were *sent*, but it gives no assurances that any of them were actually *delivered*. The reliability of the delivery mechanism is unspecified and is an appropriate marketing distinction between systems.

4.41 `mkdir` — Make directories

4.41.1 Synopsis

```
mkdir [-p] [-m mode] dir...
```

4.41.2 Description

The `mkdir` utility shall create the directories specified by the operands, in the order specified.

For each *dir* operand, the `mkdir` utility shall perform actions equivalent to the POSIX.1 {8} `mkdir()` function, called with the following arguments:

- (1) The *dir* operand is used as the *path* argument.
- (2) The value of the bitwise inclusive OR of `S_IRWXU`, `S_IRWXG`, and `S_IRWXO` is used as the *mode* argument. (If the `-m` option is specified, the *mode* option-argument overrides this default.)

4.41.3 Options

The `mkdir` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`-m mode` Set the file permission bits of the newly-created directory to the specified *mode* value. The *mode* option-argument shall be the same as the *mode* operand defined for the `chmod` utility (see 4.7). In the *symbolic_mode* strings, the *op* characters `+` and `-` shall be interpreted relative to an assumed initial mode of `a=rwx`; `+` shall add permissions to the default mode, `-` shall delete permissions from the default mode.

`-p` Create any missing intermediate pathname components.

For each *dir* operand that does not name an existing directory, effects equivalent to those caused by following command shall occur:

```
mkdir -p -m $(umask -S),u+wx $(dirname dir) &&
mkdir [-m mode] dir
```

where the `[-m mode]` option represents that option supplied to the original invocation of `mkdir`, if any.

Each *dir* operand that names an existing directory shall be ignored without error.

4.41.4 Operands

The following operand shall be supported by the implementation:

dir A pathname of a directory to be created.

4.41.5 External Influences

4.41.5.1 Standard Input

None.

4.41.5.2 Input Files

None.

4.41.5.3 Environment Variables

The following environment variables shall affect the execution of `mkdir`:

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

8781	LANG	This variable shall determine the locale to use for the
8782		locale categories when both LC_ALL and the correspond-
8783		ing environment variable (beginning with LC_) do not
8784		specify a locale. See 2.6.
8785	LC_ALL	This variable shall determine the locale to be used to over-
8786		ride any values for locale categories specified by the set-
8787		tings of LANG or any environment variables beginning
8788		with LC_ .
8789	LC_CTYPE	This variable shall determine the locale for the interpreta-
8790		tion of sequences of bytes of text data as characters (e.g.,
8791		single- versus multibyte characters in arguments).
8792	LC_MESSAGES	This variable shall determine the language in which mes-
8793		sages should be written.

8794 **4.41.5.4 Asynchronous Events**

8795 Default.

8796 **4.41.6 External Effects**

8797 **4.41.6.1 Standard Output**

8798 None.

8799 **4.41.6.2 Standard Error**

8800 Used only for diagnostic messages.

8801 **4.41.6.3 Output Files**

8802 None.

8803 **4.41.7 Extended Description**

8804 None.

8805 **4.41.8 Exit Status**

8806 The `mkdir` utility shall exit with one of the following values:

8807	0	All the specified directories were created successfully or the <code>-p</code> option
8808		was specified and all the specified directories now exist.
8809	>0	An error occurred.

4.41.9 Consequences of Errors

Default.

4.41.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The default file mode for directories is `a=rwx` (777) with selected permissions removed in accordance with the file mode creation mask. For intermediate path name components created by `mkdir`, the mode is the default modified by `u+w` so that the subdirectories can always be created regardless of the file mode creation mask; if different ultimate permissions are desired for the intermediate directories, they can be changed afterward with `chmod`.

Application writers should note that some of the requested directories may have been created even if an error occurs.

History of Decisions Made

The System V `-m` option was added to control the file mode.

The System V `-p` option was added to create any needed intermediate directories, to complement the functionality provided `rmdir` for removing directories in the path prefix as they become empty. Because no error is produced if any path component already exists, the `-p` option is also useful to ensure that a particular directory exists.

The functionality of `mkdir` is described substantially through a reference to the `mkdir()` function in POSIX.1 {8}. For example, by default, the mode of the directory is affected by the file mode creation mask in accordance with the specified behavior of POSIX.1 {8} `mkdir()`. In this way, there is less duplication of effort required for describing details of the directory creation.

8834 **4.42 mkfifo — Make FIFO special files**

8835 **4.42.1 Synopsis**

8836 `mkfifo [-m mode] file ...`

8837 **4.42.2 Description**

8838 The `mkfifo` utility shall create the FIFO special files specified by the operands, in
8839 the order specified.

8840 For each *file* operand, the `mkfifo` utility shall perform actions equivalent to the
8841 POSIX.1 {8} `mkfifo()` function, called with the following arguments:

- 8842 (1) The *file* operand is used as the *path* argument.
- 8843 (2) The value of the bitwise inclusive OR of `S_IRUSR`, `S_IWUSR`, `S_IRGRP`,
8844 `S_IWGRP`, `S_IROTH`, and `S_IWOTH` is used as the *mode* argument. (If the
8845 `-m` option is specified, the *mode* option-argument overrides this default.)

8846 **4.42.3 Options**

8847 The `mkfifo` utility shall conform to the utility argument syntax guidelines
8848 described in 2.10.2.

8849 The following option shall be supported by the implementation:

8850 `-m mode` Set the file permission bits of the newly-created FIFO to the
8851 specified *mode* value. The *mode* option-argument shall be the
8852 same as the *mode* operand defined for the `chmod` utility (see 4.7).
8853 In the *symbolic_mode* strings, the *op* characters `+` and `-` shall be
8854 interpreted relative to an assumed initial mode of `a=rw`.

8855 **4.42.4 Operands**

8856 The following operand shall be supported by the implementation:

8857 *file* A pathname of the FIFO special file to be created.

8858 **4.42.5 External Influences**

8859 **4.42.5.1 Standard Input**

8860 None.

4.42.5.2 Input Files

None.

4.42.5.3 Environment Variables

The following environment variables shall affect the execution of `mkfifo`:

LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).
LC_MESSAGES	This variable shall determine the language in which messages should be written.

4.42.5.4 Asynchronous Events

Default.

4.42.6 External Effects**4.42.6.1 Standard Output**

None.

4.42.6.2 Standard Error

Used only for diagnostic messages.

4.42.6.3 Output Files

None.

4.42.7 Extended Description

None.

4.42.8 Exit Status

The `mkfifo` utility shall exit with one of the following values:

- 0 All the specified FIFO special files were created successfully.
- >0 An error occurred.

4.42.9 Consequences of Errors

Default.

4.42.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

None.

History of Decisions Made

This new utility was added to permit shell applications to create FIFO special files.

The `-m` option was added to control the file mode, for consistency with the similar functionality provided the `mkdir` utility.

Earlier drafts included a `-p` option similar to `mkdir`'s `-p` option that created intermediate directories leading up to the FIFO specified by the final component. This was removed because it is not commonly needed and is not common practice with similar utilities.

The functionality of `mkfifo` is described substantially through a reference to the `mkfifo()` function in POSIX.1. For example, by default, the mode of the FIFO file is affected by the file mode creation mask in accordance with the specified behavior of POSIX.1 {8} `mkfifo()`. In this way, there is less duplication of effort required for describing details of the file creation.

4.43 mv — Move files

4.43.1 Synopsis

mv [-fi] *source_file target_file*

mv [-fi] *source_file ... target_dir*

4.43.2 Description

In the first synopsis form, the mv utility shall move the file named by the *source_file* operand to the *destination* specified by the *target_file*. This first synopsis form is assumed when the final operand does not name an existing directory.

In the second synopsis form, mv shall move each file named by a *source_file* operand to a *destination* file in the existing directory named by the *target_dir* operand. The *destination* path for each *source_file* shall be the concatenation of the target directory, a single slash character, and the last pathname component of the *source_file*.

If any operand specifies an existing file of a type not specified by POSIX.1 {8}, the behavior is implementation defined.

This second form is assumed when the final operand names an existing directory.

For each *source_file* the following steps shall be taken:

- (1) If the destination path exists, the -f option is not specified, and either of the following conditions is true:

- (a) The permissions of the destination path do not permit writing and the standard input is a terminal.

- (b) The -i option is specified.

the mv utility shall write a prompt to standard error and read a line from standard input. If the response is not affirmative, mv shall do nothing more with the current *source_file* and go on to any remaining *source_files*.

- (2) The mv utility shall perform actions equivalent to the POSIX.1 {8} *rename()* function, called with the following arguments:

- (a) The *source_file* operand is used as the *old* argument.

- (b) The destination path is used as the *new* argument.

If this succeeds, mv shall do nothing more with the current *source_file* and go on to any remaining *source_files*. If this fails for any reasons other than those described for the *errno* [EXDEV] in POSIX.1 {8}, mv shall write a diagnostic message to standard error, do nothing more with the current *source_file*, and go on to any remaining *source_files*.

8947 (3) If the destination path exists, and it is a file of type directory and
 8948 *source_file* is not a file of type directory, or it is a file not of type directory
 8949 and *source_file* is a file of type directory, *mv* shall write a diagnostic mes-
 8950 sage to standard error, do nothing more with the current *source_file*, and
 8951 go on to any remaining *source_files*.

8952 (4) If the destination path exists, *mv* shall attempt to remove it. If this fails
 8953 for any reason, *mv* shall write a diagnostic message to standard error, do
 8954 nothing more with the current *source_file*, and go on to any remaining
 8955 *source_files*.

8956 (5) The file hierarchy rooted in *source_file* shall be duplicated as a file hierar-
 8957 chy rooted in the destination path. The following characteristics of each
 8958 file in the file hierarchy shall be duplicated:

8959 (a) The time of last data modification and time of last access.

8960 (b) The user ID and group ID.

8961 (c) The file mode.

8962 If the user ID, group ID, or file mode of a regular file cannot be dupli-
 8963 cated, the file mode bits *S_ISUID* and *S_ISGID* shall not be duplicated.

8964 When files are duplicated to another file system, the implementation may 1
 8965 require that the process invoking *mv* have read access to each file being 1
 8966 duplicated. 1

8967 If the duplication of the file hierarchy fails for any reason, *mv* shall write
 8968 a diagnostic message to standard error, do nothing more with the current
 8969 *source_file*, and go on to any remaining *source_files*.

8970 If the duplication of the file characteristics fails for any reason, *mv* shall
 8971 write a diagnostic message to standard error, but this failure shall not
 8972 cause *mv* to modify its exit status.

8973 (6) The file hierarchy rooted in *source_file* shall be removed. If this fails for
 8974 any reason, *mv* shall write a diagnostic message to the standard error, do
 8975 nothing more with the current *source_file*, and go on to any remaining
 8976 *source_files*.

8977 4.43.3 Options

8978 The *mv* utility shall conform to the utility argument syntax guidelines described
 8979 in 2.10.2.

8980 The following options shall be supported by the implementation:

8981 *-f* Do not prompt for confirmation if the *destination* path exists.
 8982 Any previous occurrences of the *-i* option shall be ignored.

8983 *-i* Prompt for confirmation if the destination path exists. Any previ-
8984 ous occurrences of the *-f* option shall be ignored.

8985 Specifying more than one of the *-f* or *-i* options shall not be considered an error.
8986 The last option specified shall determine *mv*'s behavior.

8987 **4.43.4 Operands**

8988 The following operands shall be supported by the implementation:

8989	<i>source_file</i>	A pathname of a file or directory to be moved.
8990	<i>target_file</i>	A new pathname for the file or directory being moved.
8991	<i>target_dir</i>	A pathname of an existing directory into which to move the input
8992		files.

8993 **4.43.5 External Influences**

8994 **4.43.5.1 Standard Input**

8995 Used to read an input line in response to each prompt specified in Standard
8996 Error. 4.43.6.2. Otherwise, the standard input shall not be used.

8997 **4.43.5.2 Input Files**

8998 The input files specified by each *source_file* operand can be of any file type.

8999 **4.43.5.3 Environment Variables**

9000 The following environment variables shall affect the execution of *mv*:

9001	LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
9002		
9003		
9004		
9005	LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
9006		
9007		
9008		
9009	LC_COLLATE	This variable shall determine the locale for the behavior of ranges, equivalence classes, and multicharacter collating elements used in the extended regular expression defined for the <i>yesexpr</i> locale keyword in the LC_MESSAGES category.
9010		
9011		
9012		
9013		

9014 **LC_CTYPE** This variable shall determine the locale for the interpreta-
 9015 tion of sequences of bytes of text data as characters (e.g.,
 9016 single- versus multibyte characters in arguments) and the
 9017 behavior of character classes within regular expressions
 9018 used in the extended regular expression defined for the
 9019 `yesexpr` locale keyword in the `LC_MESSAGES` category.

9020 **LC_MESSAGES** This variable shall determine the processing of affirmative
 9021 responses and the language in which messages should be
 9022 written.

9023 **4.43.5.4 Asynchronous Events**

9024 Default.

9025 **4.43.6 External Effects**

9026 **4.43.6.1 Standard Output**

9027 None.

9028 **4.43.6.2 Standard Error**

9029 Prompts shall be written to the standard error under the conditions specified in
 9030 4.43.2. The prompts shall contain the *destination* pathname, but their format is
 9031 otherwise unspecified. Otherwise, the standard error shall be used only for diag-
 9032 nostic messages.

9033 **4.43.6.3 Output Files**

9034 The output files may be of any file type.

9035 **4.43.7 Extended Description**

9036 None.

9037 **4.43.8 Exit Status**

9038 The `mv` utility shall exit with one of the following values:

9039 0 All input files were moved successfully.

9040 >0 An error occurred.

4.43.9 Consequences of Errors

If the copying or removal of *source_file* is prematurely terminated by a signal or error, `mv` may leave a partial copy of *source_file* at the source or destination. The `mv` utility shall not modify both *source_file* and the destination path simultaneously; termination at any point shall leave either *source_file* or the destination path complete.

4.43.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

If the current directory contains only files *a* (of any type defined by POSIX.1 {8}), *b* (also of any type), and a directory *c*:

```
mv a b c
mv c d
```

will result with the original files *a* and *b* residing in the directory *d* in the current directory.

History of Decisions Made

Previous versions of this draft diverged from *SVID* and BSD historical practice in that they required that when the destination path exists, the `-f` option is not specified, and input is not a terminal, `mv` shall fail. This was done for compatibility with `cp`. This draft returns to historical practice. It should be noted that this is consistent with the POSIX.1 {8} function *rename()*, which does not require write permission on the target.

For absolute clarity, paragraph (1), describing `mv`'s behavior when prompting for confirmation, should be interpreted in the following manner:

```
if (exists AND (NOT f_option) AND
    ((not_writable AND input_is_terminal) OR i_option))
```

The `-i` option exists on BSD systems, giving applications and users a way to avoid accidentally unlinking files when moving others. When the standard input is not a terminal, the 4.3BSD `mv` deletes all existing destination paths without prompting, even when `-i` is specified; this is inconsistent with the behavior of the 4.3BSD `cp` utility, which always generates an error when the file is unwritable and the standard input is not a terminal. The working group decided that use of `-i` is a request for interaction, so when the *destination* path exists, the utility takes instructions from whatever responds to standard input.

The *rename()* function is able to move directories within the same file system. Some historical versions of `mv` have been able to move directories, but not to a different file system. The working group felt that this was an annoying inconsistency, so the standard requires directories to be movable even across file systems. There is no `-R` option to confirm that moving a directory is actually intended, since such an option was not required for moving directories in historical practice. Requiring the application to specify it sometimes, depending on the

destination, seemed just as inconsistent. The semantics of the *rename()* function were preserved as much as possible. For example, *mv* is not permitted to “rename” files to or from directories, even though they might be empty and removable.

Historic implementations of *mv* did not exit with a nonzero exit status if they were unable to duplicate any file characteristics when moving a file across file systems, nor did they write a diagnostic message for the user. The former behavior has been preserved to prevent scripts from breaking; a diagnostic message is now required, however, so that users are alerted that the file characteristics have changed.

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified, because implementations may desire more descriptive prompts than those used on historical implementations. Therefore, an application not using the *-f* option or using the *-i* option relies on the system to provide the most suitable dialogue directly with the user, based on the behavior specified.

4.44 *nohup* — Invoke a utility immune to hangups

4.44.1 Synopsis

nohup utility [argument ...]

4.44.2 Description

The *nohup* utility shall invoke the utility named by the *utility* operand with arguments supplied as the *argument* operands. At the time the named *utility* is invoked, the SIGHUP signal shall be set to be ignored.

If the standard output is a terminal, all output written by the named *utility* to its standard output shall be appended to the end of the file *nohup.out* in the current directory. If *nohup.out* cannot be created or opened for appending, the output shall be appended to the end of the file *nohup.out* in the directory specified by the **HOME** environment variable. If neither file can be created or opened for appending, *utility* shall not be invoked. If a file is created, the file's permission bits shall be set to S_IRUSR | S_IWUSR instead of the default specified in 2.9.1.4.

If the standard error is a terminal, all output written by the named *utility* to its standard error shall be redirected to the same file descriptor as the standard output.

9115 4.44.3 Options

9116 None.

9117 4.44.4 Operands

9118 The following operands shall be supported by the implementation:

9119 *utility* The name of a utility that is to be invoked. If the *utility* operand
9120 names any of the special built-in utilities in 3.14, the results are
9121 undefined.

9122 *argument* Any string to be supplied as an argument when invoking the util-
9123 ity named by the *utility* operand.

9124 4.44.5 External Influences

9125 4.44.5.1 Standard Input

9126 None.

9127 4.44.5.2 Input Files

9128 None.

9129 4.44.5.3 Environment Variables

9130 The following environment variables shall affect the execution of `nohup`:

9131 **HOME** This variable shall determine the pathname of the user's
9132 home directory: if the output file `nohup.out` cannot be
9133 created in the current directory, the `nohup` utility shall
9134 use the directory named by **HOME** to create the file.

9135 **LANG** This variable shall determine the locale to use for the
9136 locale categories when both **LC_ALL** and the correspond-
9137 ing environment variable (beginning with **LC_**) do not
9138 specify a locale. See 2.6.

9139 **LC_ALL** This variable shall determine the locale to be used to over-
9140 ride any values for locale categories specified by the set-
9141 tings of **LANG** or any environment variables beginning
9142 with **LC_**.

9143 **LC_CTYPE** This variable shall determine the locale for the interpreta-
9144 tion of sequences of bytes of text data as characters (e.g.,
9145 single- versus multibyte characters in arguments).

9146 **LC_MESSAGES** This variable shall determine the language in which mes-
 9147 sages should be written.

9148 **PATH** This variable shall determine the search path that shall
 9149 be used to locate the utility to be invoked. See 2.6.

9150 **4.44.5.4 Asynchronous Events**

9151 The `nohup` utility shall take the standard action for all signals (see 2.11.5.4),
 9152 except that `SIGHUP` shall be ignored.

9153 **4.44.6 External Effects**

9154 **4.44.6.1 Standard Output**

9155 If the standard output is not a terminal, the standard output of `nohup` shall be
 9156 the standard output generated by the execution of the *utility* specified by the
 9157 operands. Otherwise, nothing shall be written to the standard output.

9158 **4.44.6.2 Standard Error**

9159 If the standard output is a terminal, a message shall be written to the standard
 9160 error, indicating the name of the file to which the output is being appended. The
 9161 name of the file shall be either `nohup.out` or `$HOME/nohup.out`.

9162 **4.44.6.3 Output Files**

9163 If the standard output is a terminal, all output written by the named *utility* to the
 9164 standard output and standard error is appended to the file `nohup.out`, which is
 9165 created if it does not already exist.

9166 **4.44.7 Extended Description**

9167 None.

9168 **4.44.8 Exit Status**

9169 The `nohup` utility shall exit with one of the following values:

9170	126	The utility specified by <i>utility</i> was found but could not be invoked.	1
9171	127	An error occurred in the <code>nohup</code> utility or the utility specified by <i>utility</i>	1
9172		could not be found.	1

9173 Otherwise, the exit status of `nohup` shall be that of the utility specified by the
 9174 *utility* operand.

4.44.9 Consequences of Errors

Default.

4.44.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

It is frequently desirable to apply `nohup` to pipelines or lists of commands. This can be done by placing pipelines and command lists in a single file; this file can then be invoked as a utility, and the `nohup` applies to everything in the file.

Alternatively, the following command can be used to apply `nohup` to a complex command:

```
nohup sh -c 'complex-command-line'
```

The 4.3BSD version ignores `SIGTERM` and `SIGHUP`, and if `./nohup.out` cannot be used, it fails instead of trying to use `$HOME/nohup.out`.

The `command`, `env`, `nohup`, and `xargs` utilities have been specified to use exit code 127 if an error occurs so that applications can distinguish “failure to find a utility” from “invoked utility exited with an error indication.” The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for “normal error conditions” and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with `[ENOENT]`, and uses 126 when any attempt to *exec* the utility fails for any other reason.

History of Decisions Made

The `cs`h utility has a built-in version of `nohup` that acts differently than this.

The term *utility* is used, rather than *command*, to highlight the fact that shell compound commands, pipelines, special built-ins, etc., cannot be used directly. However, *utility* includes user application programs and shell scripts, not just the standard utilities.

Historical versions of the `nohup` utility use default file creation semantics. Some more recent versions use the permissions specified here as an added security precaution.

Some historical implementations ignore `SIGQUIT` in addition to `SIGHUP`; others ignore `SIGTERM`. An earlier draft allowed, but did not require, `SIGQUIT` to be ignored. Several members of the balloting group objected, saying that `nohup` should only modify the handling of `SIGHUP` as required by this specification.

4.45 `od` — Dump files in various formats

4.45.1 Synopsis

`od [-v] [-A address_base] [-j skip] [-N count] [-t type_string] ... [file ...]`

4.45.2 Description

The `od` utility shall write the contents of its input files to standard output in a user-specified format.

4.45.3 Options

The `od` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that the order of presentation of the `-t` options is significant.

The following options shall be supported by the implementation:

`-A address_base`

Specify the input offset base (see 4.45.7). The *address_base* option argument shall be a character. The characters `d`, `o`, and `x` shall specify that the offset base shall be written in decimal, octal, or hexadecimal, respectively. The character `n` shall specify that the offset shall not be written.

`-j skip`

Jump over *skip* bytes from the beginning of the input. The `od` utility shall read or seek past the first *skip* bytes in the concatenated input files. If the combined input is not at least *skip* bytes long, the `od` utility shall write a diagnostic message to standard error and exit with a nonzero exit status.

By default, the *skip* option-argument shall be interpreted as a decimal number. With a leading `0x` or `0X`, the offset shall be interpreted as a hexadecimal number; otherwise, with a leading `0`, the offset shall be interpreted as an octal number. Appending the character `b`, `k`, or `m` to offset shall cause it to be interpreted as a multiple of 512, 1024, or 1 048 576 bytes, respectively.

`-N count`

Format no more than *count* bytes of input. By default, *count* shall be interpreted as a decimal number. With a leading `0x` or `0X`, *count* shall be interpreted as a hexadecimal number; otherwise, with a leading `0`, it shall be interpreted as an octal number. If *count* bytes of input (after successfully skipping, if `-j skip` is specified) are not available, it shall not be considered an error; the `od` utility shall format the input that is available.

9245 **-t** *type_string*
 9246 Specify one or more output types (see 4.45.7). The *type_string*
 9247 option-argument shall be a string specifying the types to be used
 9248 when writing the input data. The string shall consist of the type
 9249 specification characters a, c, d, f, o, u, and x, specifying named
 9250 character, character, signed decimal, floating point, octal,
 9251 unsigned decimal, and hexadecimal, respectively. The type
 9252 specification characters d, f, o, u, and x can be followed by an
 9253 optional unsigned decimal integer that specifies the number of
 9254 bytes to be transformed by each instance of the output type. The
 9255 type specification character f can be followed by an optional F, D,
 9256 or L indicating that the conversion should be applied to an item of
 9257 type *float*, *double*, or *long double*, respectively. The type
 9258 specification characters d, o, u, and x can be followed by an
 9259 optional C, S, I, or L indicating that the conversion should be
 9260 applied to an item of type *char*, *short*, *int*, or *long*, respectively.
 9261 Multiple types can be concatenated within the same *type_string*
 9262 and multiple **-t** options can be specified. Output lines shall be
 9263 written for each type specified in the order in which the type
 9264 specification characters are specified.

9265 **-v** Write all input data. Without the **-v** option, any number of
 9266 groups of output lines, which would be identical to the immedi-
 9267 ately preceding group of output lines (except for the byte offsets),
 9268 shall be replaced with a line containing only an asterisk (*).

9269 **4.45.4 Operands**

9270 The following operands shall be supported by the implementation:

9271 *file* A pathname of a file to be written. If no file operands are
 9272 specified, the standard input shall be used. The results are
 9273 unspecified if the first character of *file* is a plus-sign (+) or the
 9274 first character of the first file operand is numeric, unless at least
 9275 one of the **-A**, **-j**, **-N**, or **-t** options is specified.

9276 **4.45.5 External Influences**

9277 **4.45.5.1 Standard Input**

9278 The standard input shall be used only if no *file* operands are specified. See Input
 9279 Files.

9280 **4.45.5.2 Input Files**

9281 The input files can be any file type.

9282 **4.45.5.3 Environment Variables**

9283 The following environment variables shall affect the execution of `od`:

9284	LANG	This variable shall determine the locale to use for the
9285		locale categories when both LC_ALL and the correspond-
9286		ing environment variable (beginning with LC_) do not
9287		specify a locale. See 2.6.
9288	LC_ALL	This variable shall determine the locale to be used to over-
9289		ride any values for locale categories specified by the set-
9290		tings of LANG or any environment variables beginning
9291		with LC_ .
9292	LC_CTYPE	This variable shall determine the locale for the interpreta-
9293		tion of sequences of bytes of text data as characters (e.g.,
9294		single- versus multibyte characters in arguments and
9295		input files).
9296	LC_MESSAGES	This variable shall determine the language in which mes-
9297		sages should be written.
9298	LC_NUMERIC	This variable shall determine the locale for selecting the
9299		radix character used when writing floating-point format-
9300		ted output.

9301 **4.45.5.4 Asynchronous Events**

9302 Default.

9303 **4.45.6 External Effects**

9304 **4.45.6.1 Standard Output**

9305 See 4.45.7.

9306 **4.45.6.2 Standard Error**

9307 Used only for diagnostic messages.

2

9308 **4.45.6.3 Output Files**

9309 None.

4.45.7 Extended Description

The `od` utility shall copy sequentially each input file to standard output, transforming the input data according to the output types specified by the `-t` option(s). If no output type is specified, the default output shall be as if `-t o2` had been specified.

The number of bytes transformed by the output type specifier `c` may be variable depending on the `LC_CTYPE` category.

The default number of bytes transformed by output type specifiers `d`, `f`, `o`, `u`, and `x` shall correspond to the various C-language types as follows. If the `c89` compiler is present on the system, these specifiers shall correspond to the sizes used by default in that compiler. Otherwise, these sizes are implementation defined.

- For the type specifier characters `d`, `o`, `u`, and `x`, the default number of bytes shall correspond to the size of the underlying implementation's basic integral data type. For these specifier characters, the implementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-language types *char*, *short*, *int*, and *long*. These numbers can also be specified by an application as the characters `C`, `S`, `I`, and `L`, respectively. The byte order used when interpreting numeric values is implementation defined, but shall correspond to the order in which a constant of the corresponding type is stored in memory on the system.

- For the type specifier character `f`, the default number of bytes shall correspond to the number of bytes in the underlying implementation's basic double precision floating point data type. The implementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-language types *float*, *double*, and *long double*. These numbers can also be specified by an application as the characters `F`, `D`, and `L`, respectively.

The type specifier character `a` specifies that bytes shall be interpreted as named characters from the International Reference Version (IRV) of ISO/IEC 646 {1}. Only the least significant seven bits of each byte shall be used for this type specification. Bytes with the values listed in Table 4-8 shall be written using the corresponding names for those characters.

The type specifier character `c` specifies that bytes shall be interpreted as characters specified by the current setting of the `LC_CTYPE` locale category. Characters listed in Table 2-15 (see 2.12) shall be written as the corresponding escape sequences, except that backslash shall be written as a single backslash and a NUL shall be written as `\0`. Other nonprintable characters shall be written as one three-digit octal number for each byte in the character. If the size of a byte on the system is greater than nine bits, the format used for nonprintable characters is implementation-defined. Printable multibyte characters shall be written in the area corresponding to the first byte of the character; the two-character sequence `**` shall be written in the area corresponding to each remaining byte in the character, as an indication that the character is continued.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

Table 4-8 – od Named Characters

<u>Value</u>	<u>Name</u>	<u>Value</u>	<u>Name</u>	<u>Value</u>	<u>Name</u>	<u>Value</u>	<u>Name</u>
\000	nul	\001	soh	\002	stx	\003	etx
\004	eot	\005	enq	\006	ack	\007	bel
\010	bs	\011	ht	\012	lf or nl*	\013	vt
\014	ff	\015	cr	\016	so	\017	si
\020	dle	\021	dc1	\022	dc2	\023	dc3
\024	dc4	\025	nak	\026	syn	\027	etb
\030	can	\031	em	\032	sub	\033	esc
\034	fs	\035	gs	\036	rs	\037	us
\040	sp	\177	del				

NOTE: The \012 value may be written either as lf or nl.

The input data shall be manipulated in blocks, where a block is defined as a multiple of the least common multiple of the number of bytes transformed by the specified output types. If the least common multiple is greater than 16, the results are unspecified. Each input block shall be written as transformed by each output type, one per written line, in the order that the output types were specified. If the input block size is larger than the number of bytes transformed by the output type, the output type shall sequentially transform the parts of the input block and the output from each of the transformations shall be separated by one or more <blank>s.

If, as a result of the specification of the `-N` option or end-of-file being reached on the last input file, input data only partially satisfies an output type, the input shall be extended sufficiently with null bytes to write the last byte of the input.

Unless `-A n` is specified, the first output line produced for each input block shall be preceded by the input offset, cumulative across input files, of the next byte to be written. The format of the input offset is unspecified; however, it shall not contain any <blank>s, shall start at the first character of the output line, and shall be followed by one or more <blank>s. In addition, the offset of the byte following the last byte written shall be written after all the input data has been processed, but shall not be followed by any <blank>s.

If no `-A` option is specified, the input offset base is unspecified.

4.45.8 Exit Status

The `od` utility shall exit with one of the following values:

- 0 All input files were processed successfully.
- >0 An error occurred.

4.45.9 Consequences of Errors

Default.

4.45.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

If a file containing 128 bytes with decimal values zero through 127, in increasing order, is supplied as standard input to the command:

```
od -A d -t a
```

on an implementation using an input block size of 16 bytes, the standard output, independent of the current locale setting, would be similar to:

```
0000000 nul soh stx etx eot enq ack bel  bs  ht  nl  vt  ff  cr  so  si
0000016 dle dc1 dc2 dc3 dc4 nak syn etb can  em sub esc  fs  gs  rs  us
0000032 sp  !  "  #  $  %  &  '  (  )  *  +  ,  -  .  /
0000048 0  1  2  3  4  5  6  7  8  9  :  ;  <  =  >  ?
0000064 @  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O
0000080 P  Q  R  S  T  U  V  W  X  Y  Z  [  \  ]  ^  _
0000096 `  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o
0000112 p  q  r  s  t  u  v  w  x  y  z  {  |  }  ~ del
0000128
```

Note that this standard allows `nl` or `lf` to be used as the name for the ISO/IEC 646 {1} IRV character with decimal value 10. The IRV names this character `lf` (line feed), but traditional implementations on which POSIX.2 are based have referred to this character as newline (`nl`) and the POSIX Locale character set symbolic name for the corresponding character is `<newline>`.

The command:

```
od -A o -t o2x2x -n 18
```

on a system with 32-bit words and an implementation using an input block size of 16 bytes could write 18 bytes in approximately the following format:

```
0000000 032056 031440 041123 042040 052516 044530 020043 031464
          342e  3320  4253  4420  554e  4958  2023  3334
          342e3320      42534420      554e4958      20233334
0000020 032472
          353a
          353a0000
0000022
```

The command:

```
od -A d -t f -t o4 -t x4 -n 24 -j 0x15
```

on a system with 64-bit doubles (for example, the IEEE Std 754 double precision floating point format) would skip 21 bytes of input data and then write 24 bytes in approximately the following format:


```

9431      00000000      1.0000000000000000e+00      1.5735000000000000e+01
9432                07774000000 000000000000 10013674121 35341217270
9433                3ff000000 000000000 402f7851 eb851eb8
9434      0000016      1.4066823000000000e+02
9435                10030312542 04370303230
9436                40619562 23e18698
9437      0000024

```

9438 History of Decisions Made

9439 The `od` utility has gone through several names in previous drafts, including `hd`,
 9440 `xd`, and most recently `hexdump`. There were several objections to all of these
 9441 based on the following reasons:

- 9442 — The `hd` and `xd` names conflicted with existing utilities that behaved dif-
 9443 ferently.
- 9444 — The `hexdump` description was much more complex than needed for a simple
 9445 dump utility.
- 9446 — The `od` utility has been available on all traditional implementations and
 9447 there was no need to create a new name for a utility so similar to the exist-
 9448 ing `od` utility.

9449 The original reasons for not standardizing historical `od` were also fairly
 9450 widespread. Those reasons are given below along with rationale explaining why
 9451 the developers of this standard believe that this version does not suffer from the
 9452 indicated problem:

- 9453 — The BSD and System V versions of `od` have diverged and the intersection of
 9454 features provided by both does not meet the needs of the user community.
 9455 In fact, the System V version only provides a mechanism for dumping octal
 9456 bytes and *shorts*, signed and unsigned decimal *shorts*, hexadecimal *shorts*,
 9457 and ASCII characters. BSD added the ability to dump *floats*, *doubles*,
 9458 named ASCII characters, and octal, signed decimal, unsigned decimal, and
 9459 hexadecimal *longs*. The version presented here provides more normalized
 9460 forms for dumping bytes, *shorts*, *ints*, and *longs* in octal, signed decimal,
 9461 unsigned decimal, and hexadecimal; *float*, *double*, and *long double*; and
 9462 named ASCII as well as current locale characters.
- 9463 — It would not be possible to come up with a compatible superset of the BSD
 9464 and System V flags that met the requirements of this standard. The histor-
 9465 ical default `od` output is the specified default output of this utility. None of
 9466 the option letters chosen for this version of `od` conflict with any of the
 9467 options to historical versions of `od`.
- 9468 — On systems with different sizes for *short*, *int*, and *long*, there was no way to
 9469 ask for dumps of *ints*, even in the BSD version. The way options are
 9470 named, there is no easy way to extend the namespace for these problems.
 9471 This is why the `-t` option was added with type specifiers more closely
 9472 matched to the `printf()` formats used in the rest of this standard and the
 9473 optional field sizes were added to the `d`, `f`, `o`, `u`, and `x` type specifiers. It is
 9474 also one of the reasons why the historical practice was not mandated as a

required obsolescent form of `od`. (Although the old versions of `od` are not listed as an obsolescent form, implementations are urged to continue to recognize the old forms they have recognized for a few years.) The `a`, `c`, `f`, `o`, and `x` types match the meaning of the corresponding format characters in the historical implementations of `od` except for the default sizes of the fields converted. The `d` format is signed in this specification to match the `printf()` notation. (Historical versions of `od` used `d` as a synonym for `u` in this version. The System V implementation uses `s` for signed decimal; BSD uses `i` for signed decimal and `s` for null terminated strings.) Other than `d` and `u`, all of the type specifiers match format characters in the historical BSD version of `od`.

The sizes of the C-language types *char*, *short*, *int*, *long*, *float*, *double*, and *long double* are used even though it is recognized that there may be zero or more than one compiler for the C language on an implementation and that they may use different sizes for some of these types. [For example, one compiler might use 2-byte *shorts*, 2-byte *ints*, and 4-byte *longs* while another compiler (or an option to the same compiler) uses 2-byte *shorts*, 4-byte *ints*, and 4-byte *longs*.] Nonetheless, there has to be a basic size known by the implementation for these types, corresponding to the values reported by invocations of the `getconf` utility (see 4.26) when called with *system_var* operands `UCHAR_MAX`, `USHORT_MAX`, `UINT_MAX`, and `ULONG_MAX` for the types *char*, *short*, *int*, and *long*, respectively. There are similar constants required by the C Standard {7}, but not required by POSIX.1 {8} or POSIX.2. They are `FLT_MANT_DIG`, `DBL_MANT_DIG`, and `LDBL_MANT_DIG` for the types *float*, *double*, and *long double*, respectively. If the optional `c89` utility (see A.1) is provided by the implementation and used as specified by this standard, these are the sizes that would be provided. If an option is used that specifies different sizes for these types, there is no guarantee that the `od` utility will be able to correctly interpret binary data output by such a program.

POSIX.2 requires that the numeric values of these lengths be recognized by the `od` utility and that symbolic forms also be recognized. Thus a portable application can always look at an array of *unsigned long* data elements using `od -t uL`.

- The method of specifying the format for the address field based on specifying a starting offset in a file unnecessarily tied the two together. The `-A` option now specifies the address base and the `-S` option specifies a starting offset. Applications are warned not to use filenames starting with `+` or a first operand starting with a numeric character so that the old functionality can be maintained by implementations, unless they specify one of the new options specified by POSIX.2. To guarantee that one of these filenames will always be interpreted as a file name, an application could always specify the address base format with the `-A` option.
- It would be hard to break the dependence on US ASCII to get an internationalized utility. It does not seem to be any harder for `od` to dump characters in the current locale than it is for the `ed` or `sed` `l` commands. The `c`

9521 type specifier does this with no problem and is completely compatible with
 9522 the historical implementations of the `c` format character when the current
 9523 locale uses a superset of ISO/IEC 646 {1} as a code set. The `a` type specifier
 9524 (from the BSD `a` format character) was left as a portable means to dump
 9525 ASCII [or more correctly ISO/IEC 646 {1} (IRV)] so that headers produced by
 9526 `pax` could be deciphered even on systems that do not use ISO/IEC 646 {1} as
 9527 a subset of their base code set.

9528 The use of `**` as an indication of continuation of a multibyte character in `c`
 9529 specifier output was chosen based on seeing an implementation that uses this
 9530 method. The continuation bytes have to be marked in a way that will not be
 9531 ambiguous with another single- or multibyte character.

9532 An earlier draft used `-S` and `-n`, respectively, for the `-j` and `-N` options in this
 9533 draft. These were changed to avoid conflicts with historical implementations.

9534 **4.46 `paste` — Merge corresponding or subsequent lines of files**

9535 **4.46.1 Synopsis**

9536 `paste [-s] [-d list] file ...`

9537 **4.46.2 Description**

9538 The `paste` utility shall concatenate the corresponding lines of the given input
 9539 files, and write the resulting lines to standard output.

9540 The default operation of `paste` shall concatenate the corresponding lines of the
 9541 input files. The `<newline>` character of every line except the line from the last
 9542 input file shall be replaced with a `<tab>` character.

9543 If an end-of-file condition is detected on one or more input files, but not all input
 9544 files, `paste` shall behave as though empty lines were read from the file(s) on
 9545 which end-of-file was detected, unless the `-s` option is specified.

9546 **4.46.3 Options**

9547 The `paste` utility shall conform to the utility argument syntax guidelines
 9548 described in 2.10.2.

9549 The following options shall be supported by the implementation:

9550	<code>-d <i>list</i></code>	Unless a backslash character appears in <i>list</i> , each character in	2
9551		<i>list</i> is an element specifying a delimiter character. If a backslash	2
9552		character appears in <i>list</i> , the backslash character and one or more	2
9553		characters following it are an element specifying a delimiter char-	2
9554		acter as described below. These elements specify one or more	2

9555 delimiters to use, instead of the default <tab>, to replace the 2
 9556 <newline> character of the input lines. The elements in *list* 2
 9557 shall be used circularly; i.e., when the list is exhausted the first 2
 9558 element from the list shall be re-used. When the *-s* option is 2
 9559 specified:

- 9560 — The last <newline> character in a file shall not be modified.
- 9561 — The delimiter shall be reset to the first element of list after
 9562 each *file* operand is processed.

9563 When the *-s* option is not specified:

- 9564 — The <newline> characters in the file specified by the last *file*
 9565 operand shall not be modified.
- 9566 — The delimiter shall be reset to the first element of list each
 9567 time a line is processed from each file.

9568 If a backslash character appears in *list*, it and the character fol-
 9569 lowing it shall be used to represent the following delimiter char-
 9570 acters:

9571	\n	<newline> character
9572	\t	<tab> character
9573	\\	backslash character
9574	\0	Empty string (not a null character). If \0 is immedi-
9575		ately followed by the character <i>x</i> , the character <i>X</i> , or
9576		any character defined by the LC_CTYPE digit keyword
9577		(see 2.5.2.1), the results are unspecified.

9578 If any other characters follow the backslash, the results are
 9579 unspecified.

9580 *-s* Concatenate all of the lines of each separate input file in com-
 9581 mand line order. The <newline> character of every line except
 9582 the last line in each input file shall be replaced with the <tab>
 9583 character, unless otherwise specified by the *-d* option.

9584 4.46.4 Operands

9585 The following operand shall be supported by the implementation:

9586	<i>file</i>	A pathname of an input file. If <i>-</i> is specified for one or more of
9587		the <i>files</i> , the standard input shall be used; the standard input
9588		shall be read one line at a time, circularly, for each instance of <i>-</i> .
9589		Implementations shall support pasting of at least 12 <i>file</i>
9590		operands.

9591 **4.46.5 External Influences**

9592 **4.46.5.1 Standard Input**

9593 The standard input shall be used only if one or more *file* operands is `-`. See Input
9594 Files.

9595 **4.46.5.2 Input Files**

9596 The input files shall be text files, except that line lengths shall be unlimited.

9597 **4.46.5.3 Environment Variables**

9598 The following environment variables shall affect the execution of `paste`:

9599	LANG	This variable shall determine the locale to use for the
9600		locale categories when both LC_ALL and the correspond-
9601		ing environment variable (beginning with LC_) do not
9602		specify a locale. See 2.6.
9603	LC_ALL	This variable shall determine the locale to be used to over-
9604		ride any values for locale categories specified by the set-
9605		tings of LANG or any environment variables beginning
9606		with LC_ .
9607	LC_CTYPE	This variable shall determine the locale for the interpreta-
9608		tion of sequences of bytes of text data as characters (e.g.,
9609		single- versus multibyte characters in arguments and
9610		input files).
9611	LC_MESSAGES	This variable shall determine the language in which mes-
9612		sages should be written.

9613 **4.46.5.4 Asynchronous Events**

9614 Default.

9615 **4.46.6 External Effects**

9616 **4.46.6.1 Standard Output**

9617 Concatenated lines of input files shall be separated by the `<tab>` character (or
9618 other characters under the control of the `-d` option) and terminated by a `<new-`
9619 `line>` character.

4.46.6.2 Standard Error

Used only for diagnostic messages.

4.46.6.3 Output Files

None.

4.46.7 Extended Description

None.

4.46.8 Exit Status

The `paste` utility shall exit with one of the following values:

0 Successful completion.

>0 An error occurred.

4.46.9 Consequences of Errors

If one or more input files cannot be opened when the `-s` option is not specified, a diagnostic message shall be written to standard error, but no output shall be written to standard output. If the `-s` option is specified, the `paste` utility shall provide the default behavior described in 2.11.9.

4.46.10 Rationale. *(This subclause is not a part of P1003.2)***Examples, Usage**

When the escape sequences of the *list* option-argument are used in a shell script, they must be quoted; otherwise, the shell treats the `\` as a special character.

Write out a directory in four columns:

```
ls | paste - - - -
```

Combine pairs of lines from a file into single lines:

```
paste -s -d "\t\n" file
```

Portable applications should only use the specific backslash escaped delimiters presented in this standard. Historical implementations treat `\x`, where `x` is not in this list, as `x`, but future implementations are free to expand this list to recognize other common escapes similar to those accepted by `printf` and other standard utilities.

Most of the standard utilities work on text files. The `cut` utility can be used to turn files with arbitrary line lengths into a set of text files containing the same

data. The `paste` utility can be used to create (or recreate) files with arbitrary line lengths. For example, if `file` contains long lines:

```
cut -b 1-500 -n file > file1
cut -b 501- -n file > file2
```

creates `file1` (a text file) with lines no longer than 500 bytes (plus the <new-line> character) and `file2` that contains the remainder of the data from `file`. (Note that `file2` will not be a text file if there are lines in `file` that are longer than 500 + {LINE_MAX} bytes.) The original file can be recreated from `file1` and `file2` using the command:

```
paste -d "\0" file1 file2 > file
```

The commands

```
paste -d "\0" ...
paste -d "" ...
```

are not necessarily equivalent; the latter is not specified by POSIX.2 and may result in an error. The construct `\0` is used to mean “no separator” because historical versions of `paste` did not follow the syntax guidelines and the command

```
paste -d "" ...
```

could not be handled properly by `getopt()`.

History of Decisions Made

Because most of the standards utilities work on text files, `cut` and `paste` are required to process lines of arbitrary length as a means of converting long lines from arbitrary sources into text files and converting processed text files back into files with arbitrary line lengths to interface with those applications that require long lines as input.

4.47 pathchk — Check pathnames

4.47.1 Synopsis

pathchk [-p] *pathname* ...

4.47.2 Description

The pathchk utility shall check that one or more pathnames are valid (i.e., they could be used to access or create a file without causing syntax errors) and portable (i.e., no filename truncation will result). More extensive portability checks are provided by the -p option.

By default, the pathchk utility shall check each component of each *pathname* operand based on the underlying file system. A diagnostic shall be written for each *pathname* operand that:

- is longer than {PATH_MAX} bytes (see Pathname Variable Values in POSIX.1 {8} 2.9.5),
- contains any component longer than {NAME_MAX} bytes in its containing directory,
- contains any component in a directory that is not searchable, or
- contains any character in any component that is not valid in its containing directory.

The format of the diagnostic message is not specified, but shall indicate the error detected and the corresponding *pathname* operand.

It shall not be considered an error if one or more components of a *pathname* operand do not exist as long as a file matching the pathname specified by the missing components could be created that does not violate any of the checks specified above.

4.47.3 Options

The pathchk utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following option shall be supported by the implementation:

- p Instead of performing checks based on the underlying file system, write a diagnostic for each *pathname* operand that:
 - is longer than {_POSIX_PATH_MAX} bytes (see Minimum Values in POSIX.1 {8} 2.9.2),
 - contains any component longer than {_POSIX_NAME_MAX} bytes, or

9708 — contains any character in any component that is not in the
 9709 portable filename character set (see 2.2.2.111).

9710 **4.47.4 Operands**

9711 The following operand shall be supported by the implementation:

9712 *pathname* A pathname to be checked.

9713 **4.47.5 External Influences**

9714 **4.47.5.1 Standard Input**

9715 None.

9716 **4.47.5.2 Input Files**

9717 None.

9718 **4.47.5.3 Environment Variables**

9719 The following environment variables shall affect the execution of `pathchk`:

9720 **LANG** This variable shall determine the locale to use for the
 9721 locale categories when both **LC_ALL** and the correspond-
 9722 ing environment variable (beginning with **LC_**) do not
 9723 specify a locale. See 2.6.

9724 **LC_ALL** This variable shall determine the locale to be used to over-
 9725 ride any values for locale categories specified by the set-
 9726 tings of **LANG** or any environment variables beginning
 9727 with **LC_**.

9728 **LC_CTYPE** This variable shall determine the locale for the interpreta-
 9729 tion of sequences of bytes of text data as characters (e.g.,
 9730 single- versus multibyte characters in arguments).

9731 **LC_MESSAGES** This variable shall determine the language in which mes-
 9732 sages should be written.

9733 **4.47.5.4 Asynchronous Events**

9734 Default.

9735 **4.47.6 External Effects**

9736 **4.47.6.1 Standard Output**

9737 None.

9738 **4.47.6.2 Standard Error**

9739 Used only for diagnostic messages.

9740 **4.47.6.3 Output Files**

9741 None.

9742 **4.47.7 Extended Description**

9743 None.

9744 **4.47.8 Exit Status**

9745 The `pathchk` utility shall exit with one of the following values:

9746 0 All *pathname* operands passed all of the checks.

9747 >0 An error occurred.

9748 **4.47.9 Consequences of Errors**

9749 Default.

9750 **4.47.10 Rationale.** *(This subclause is not a part of P1003.2)*

9751 **Examples, Usage**

9752 To verify that all pathnames in an imported data interchange archive are legiti-
9753 mate and unambiguous on the current system:

```
9754     pax -f archive | xargs pathchk
9755     if [ $? -eq 0 ]
9756     then
9757         pax -r -f archive
9758     else
9759         echo Investigate problems before importing files.
9760         exit 1
9761     fi
```

1

9762 To verify that all files in the current directory hierarchy could be moved to any
9763 POSIX.1 {8} conforming system that also supports the `pax` utility:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```

9764     find . -print | xargs pathchk -p
9765     if [ $? -eq 0 ]
9766     then
9767         pax -w -f archive .
9768     else
9769         echo Portable archive cannot be created.
9770         exit 1
9771     fi

```

9772 To verify that a user-supplied pathname names a readable file and that the appli-
 9773 cation can create a file extending the given path without truncation and without
 9774 overwriting any existing file:

```

9775     case $- in
9776         *C*)     reset="";;
9777         *)       reset="set +C"
9778                 set -C;;
9779     esac
9780     test -r "$path" && pathchk "$path.out" &&
9781         rm "$path.out" > "$path.out"
9782     if [ $? -ne 0 ]; then
9783         printf "%s: %s not found or %s.out fails \n" $0 "$path" "$path" 1
9784         creation checks.\n" $0 "$path" "$path" 1
9785         $reset # reset the noclobber option in case a trap 1
9786                 # on EXIT depends on it 1
9787         exit 1
9788     fi
9789     $reset
9790     PROCESSING < "$path" > "$path.out"

```

9791 The following assumptions are made in this example:

- 9792 (1) PROCESSING represents the code that will be used by the application to
 9793 use \$path once it is verified that \$path.out will work as intended.
- 9794 (2) The state of the *noclobber* option is unknown when this code is invoked
 9795 and should be set on exit to the state it was in when this code was
 9796 invoked. (The *reset* variable is used in this example to restore the ini-
 9797 tial state.)
- 9798 (3) Note the usage of `rm "$path.out" > "$path.out"`:
 - 9799 (a) The *pathchk* command has already verified, at this point, that
 9800 \$path.out will not be truncated.
 - 9801 (b) With the *noclobber* option set, the shell will verify that \$path.out
 9802 does not already exist before invoking *rm*.
 - 9803 (c) If the shell succeeded in creating \$path.out, *rm* will remove it so
 9804 that the application can create the file again in the PROCESSING
 9805 step.
 - 9806 (d) If the PROCESSING step wants the file to already exist when it is
 9807 invoked, the

9808 rm "\$path.out" > "\$path.out"

9809 should be replaced with

9810 > "\$path.out"

9811 which will verify that the file did not already exist, but leave
9812 \$path.out in place for use by PROCESSING.

9813 **History of Decisions Made**

9814 The pathchk utility is new, commissioned for this standard. It, along with the
9815 set -C (*noclobber*) option added to the shell, replaces the mktemp, validfnam,
9816 and create utilities that appeared in earlier drafts. All of these utilities were
9817 attempts to solve a few common problems:

- 9818 — Verify the validity (for several different definitions of “valid”) of a path-
9819 name supplied by a user, generated by an application, or imported from an
9820 external source,
- 9821 — Atomically create a file, and
- 9822 — Perform various string handling functions to generate a temporary file
9823 name.

9824 The test utility (see 4.62) can be used to determine if a given pathname names
9825 an existing file; it will not, however, give any indication of whether or not any
9826 component of the pathname was truncated in a directory where the
9827 {_POSIX_NO_TRUNC} feature (see Execution-Time Symbolic Constants for Portability
9828 Specification in POSIX.1 {8} 2.9.4) is not in effect. The pathchk utility pro-
9829 vided here does not check for file existence; it performs checks to determine if a
9830 pathname does exist or could be created with no pathname component truncation.

9831 The *noclobber* option added to the shell (see 3.14.11) can be used to atomically
9832 create a file. As with all file creation semantics in POSIX.1 {8}, it guarantees
9833 atomic creation, but still depends on applications to agree on conventions and
9834 cooperate on the use of files after they have been created. The create utility,
9835 included in one earlier draft, provided checking and atomic creation in a single
9836 invocation of the utility; these are orthogonal issues and need not be grouped into
9837 a single utility. Note that the *noclobber* option also provides a way of creating a
9838 lock for process synchronization; since it provides an atomic create, there is no
9839 race between a test for existence and the following creation if it did not exist.

9840 Having a function like *tmpnam()* in the C Standard {7} is important in many
9841 high-level languages. The shell programming language, however, has built-in
9842 string manipulation facilities, making it very easy to construct temporary file
9843 names. The names needed obviously depend on the application, but are fre-
9844 quently of a form similar to

9845 \$TMPDIR/*application_abbreviation*\$.*suffix*

9846 In cases where there is likely to be contention for a given suffix, a simple shell
9847 for or while loop can be used with the shell *noclobber* option to create a file
9848 without risk of collisions, as long as applications trying to use the same filename

namespace are cooperating on the use of files after they have been created.

4.48 pax — Portable archive interchange

4.48.1 Synopsis

```

9852 pax [-cdnv] [-f archive] [-s replstr] ... [pattern ...] 1
9853 pax -r [-cdiknuv] [-f archive] [-o options] ... [-p string] ... [-s replstr] 1
9854 ... [pattern ...] 1
9855 pax -w [-dituvX] [-b blocksize] [ [-a] [-f archive] ] [-o options] ... 1
9856 [-s replstr] ... [-x format] [file ...]
9857 pax -r -w [-diklntuvX] [-p string] ... [-s replstr] ... [file ...] directory

```

4.48.2 Description

The `pax` utility shall read, write, and write lists of the members of archive files and copy directory hierarchies. A variety of archive formats shall be supported; see the `-x format` option description under 4.48.3.

The action to be taken depends on the presence of the `-r` and `-w` options:

- (1) When neither the `-r` option nor the `-w` option is specified, `pax` shall write the names of the members of the archive file read from the standard input, with pathnames matching the specified patterns, to standard output. If a named file is of type directory, the file hierarchy rooted at that file shall be written out as well.
- (2) When the `-r` option is specified, but the `-w` option is not, `pax` shall extract the members of the archive file read from the standard input, with pathnames matching the specified patterns. If an extracted file is of type directory, the file hierarchy rooted at that file shall be extracted as well. The extracted files shall be created relative to the current file hierarchy.

The ownership, access and modification times, and file mode of the restored files are discussed under the `-p` option.
- (3) When the `-w` option is specified and the `-r` option is not, `pax` shall write the contents of the file operands to the standard output in an archive format. If no *file* operands are specified, a list of files to copy, one per line, shall be read from the standard input. A file of type directory shall include all of the files in the file hierarchy rooted at the file.
- (4) When both the `-r` and `-w` options are specified, `pax` shall copy the file operands to the destination directory.

- 9883 If no *file* operands are specified, a list of files to copy, one per line, shall
 9884 be read from the standard input. A file of type directory shall include all
 9885 of the files in the file hierarchy rooted at the file.
- 9886 The effect of the copy shall be as if the copied files were written to an
 9887 archive file and then subsequently extracted, except that there may be
 9888 hard links between the original and the copied files. If the destination
 9889 directory is a subdirectory of one of the files to be copied, the results are
 9890 unspecified. If the destination directory is a file of a type not defined by
 9891 POSIX.1 {8}, the results are implementation defined; otherwise it shall be
 9892 an error for the file named by the directory operand not to exist, not be
 9893 writable by the user, or not be a file of type directory.
- 9894 If, when the *-r* option is specified, intermediate directories are necessary to
 9895 extract an archive member, *pax* shall perform actions equivalent to the
 9896 POSIX.1 {8} *mkdir()* function, called with the following arguments:
- 9897 — The intermediate directory used as the *path* argument.
 - 9898 — The value of the bitwise inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO
 9899 as the *mode* argument.
- 9900 If any specified *pattern* or *file* operands are not matched by at least one file or
 9901 archive member, *pax* shall write a diagnostic message to standard error for each
 9902 one that did not match and exit with a nonzero exit status.
- 9903 The supported archive formats shall be automatically detected on input. The
 9904 default output archive format shall be implementation defined.
- 9905 A single archive can span multiple files. The *pax* utility shall determine, in an
 9906 implementation-defined manner, what file to read or write as the next file.
- 9907 If the selected archive format supports the specification of linked files, it shall be
 9908 an error if these files cannot be linked when the archive is extracted. Any of the 1
 9909 various names in the archive that represent a file can be used to select the file for 1
 9910 extraction. 1
- 9911 **4.48.3 Options**
- 9912 The *pax* utility shall conform to the utility argument syntax guidelines described
 9913 in 2.10.2, except that the order of presentation of the *-s* options is significant.
- 9914 The following options shall be supported by the implementation:
- 9915 *-r* Read an archive file from standard input.
 - 9916 *-w* Write files to the standard output in the specified archive format.
 - 9917 *-a* Append files to the end of the archive. It is implementation 1
 9918 defined which devices on the system support appending. Addi- 1
 9919 tional file formats unspecified by this standard may impose res- 1
 9920 trictions on appending. 1

9921	<code>-b</code> <i>blocksize</i>		1
9922		Block the output at a positive decimal integer number of bytes per	
9923		write to the archive file. Devices and archive formats may impose	
9924		restrictions on blocking. Blocking shall be automatically deter-	
9925		mined on input. Conforming POSIX.2 applications shall not	
9926		specify a <i>blocksize</i> value larger than 32256. Default blocking	1
9927		when creating archives depends on the archive format. (See the	
9928		<code>-x</code> option below.)	
9929	<code>-c</code>	Match all file or archive members except those specified by the	
9930		<i>pattern</i> or <i>file</i> operands.	
9931	<code>-d</code>	Cause files of type directory being copied or archived or archive	
9932		members of type directory being extracted to match only the file	
9933		or archive member itself and not the file hierarchy rooted at the	
9934		file.	
9935	<code>-f</code> <i>archive</i>	Specify the pathname of the input or output archive, overriding	
9936		the default standard input (when neither the <code>-r</code> option nor the <code>-w</code>	
9937		option is specified, or the <code>-r</code> option is specified and the <code>-w</code> option	
9938		is not) or standard output (when the <code>-w</code> option is specified and the	
9939		<code>-r</code> option is not).	
9940	<code>-i</code>	Interactively rename files or archive members. For each archive	
9941		member matching a <i>pattern</i> operand or file matching a <i>file</i>	
9942		operand, a prompt shall be written to the file <code>/dev/tty</code> . The	
9943		prompt shall contain the name of the file or archive member, but	
9944		the format is otherwise unspecified. A line shall then be read	
9945		from <code>/dev/tty</code> . If this line is blank, the file or archive member	1
9946		shall be skipped. If this line consists of a single period, the file or	
9947		archive member shall be processed with no modification to its	
9948		name. Otherwise, its name shall be replaced with the contents of	
9949		the line. The <code>pax</code> utility shall immediately exit with a nonzero	
9950		exit status if end-of-file is encountered when reading a response	
9951		or if <code>/dev/tty</code> cannot be opened for reading and writing.	
9952	<code>-k</code>	Prevent the overwriting of existing files.	
9953	<code>-l</code>	(The letter ell.) Link files. When both the <code>-r</code> and <code>-w</code> options are	
9954		specified, hard links shall be made between the source and desti-	
9955		nation file hierarchies whenever possible.	
9956	<code>-n</code>	Select the first archive member that matches each <i>pattern</i>	
9957		operand. No more than one archive member shall be matched for	
9958		each <i>pattern</i> (although members of type directory shall still	
9959		match the file hierarchy rooted at that file).	
9960	<code>-o</code> <i>options</i>	Provide information to the implementation to modify the algo-	1
9961		rithm for extracting or writing files that is specific to the file for-	1
9962		mat specified by <code>-x</code> . This version of this standard does not	1
9963		specify any such options and a Strictly Conforming POSIX.2 Appli-	1
9964		cation shall not use the <code>-o</code> option.	1

9965		NOTE: It is expected that future versions of POSIX.2 will offer additional file for-	1
9966		ats and this option will be used by POSIX.2 and other POSIX standards to	1
9967		specify such features as international file-name and file codeset translations,	1
9968		security, accounting, etc., related to each additional format.	1
9969	<code>-p string</code>	Specify one or more file characteristic options (privileges). The	
9970		<i>string</i> option-argument shall be a string specifying file charac-	
9971		teristics to be retained or discarded on extraction. The string	
9972		shall consist of the specification characters a, e, m, o, and p,	
9973		and/or other, implementation-defined, characters. Multiple	
9974		characteristics can be concatenated within the same string and	
9975		multiple <code>-p</code> options can be specified. The meaning of the	
9976		specification characters are as follows:	
9977	a	Do not preserve file access times.	
9978	e	Preserve the user ID, group ID, file mode bits (see	1
9979		2.2.2.60), access time, modification time, and any other,	1
9980		implementation-defined, file characteristics.	1
9981	m	Do not preserve file modification times.	
9982	o	Preserve the user ID and group ID.	
9983	p	Preserve the file mode bits. Other, implementation-	1
9984		defined file-mode attributes may be preserved.	1
9985		In the preceding list, “preserve” indicates that an attribute stored	
9986		in the archive shall be given to the extracted file, subject to the	1
9987		permissions of the invoking process; otherwise, the attribute shall	1
9988		be determined as part of the normal file creation action (see	1
9989		2.9.1.4).	1
9990		If neither the e nor the o specification character is specified, or	
9991		the user ID and group ID are not preserved for any reason, pax	
9992		shall not set the S_ISUID and S_ISGID bits of the file mode.	
9993		If the preservation of any of these items fails for any reason, pax	
9994		shall write a diagnostic message to standard error. Failure to	
9995		preserve these items shall affect the final exit status, but shall	
9996		not cause the extracted file to be deleted.	
9997		If file-characteristic letters in any of the <i>string</i> option-arguments	
9998		are duplicated or conflict with each other, the one(s) given last	
9999		shall take precedence. For example, if <code>-p eme</code> is specified, file	
10000		modification times shall be preserved.	
10001	<code>-s replstr</code>	Modify file or archive member names named by <i>pattern</i> or <i>file</i>	
10002		operands according to the substitution expression <i>replstr</i> , using	
10003		the syntax of the <i>ed</i> utility (see 4.20). The concepts of “address”	
10004		and “line” are meaningless in the context of the <i>pax</i> utility, and	
10005		shall not be supplied. The format shall be:	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

10006	<code>-s /old/new/[gp]</code>	
10007		where as in <code>ed</code> , <i>old</i> is a basic regular expression and <i>new</i> can contain an ampersand, <code>\n</code> (where <i>n</i> is a digit) backreferences, or subexpression matching. The <i>old</i> string shall also be permitted to contain <code><newline></code> characters.
10008		
10009		
10010		
10011		Any nonnull character can be used as a delimiter (<code>/</code> shown here).
10012		Multiple <code>-s</code> expressions can be specified; the expressions shall be applied in the order specified, terminating with the first successful substitution. The optional trailing <code>g</code> shall be as defined in the <code>ed</code> utility. The optional trailing <code>p</code> shall cause successful substitutions to be written to standard error. File or archive member names that substitute to the empty string shall be ignored when reading and writing archives.
10013		
10014		
10015		
10016		
10017		
10018		
10019	<code>-t</code>	Cause the access times of the archived files to be the same as they were before being read by <code>pax</code> .
10020		
10021	<code>-u</code>	Ignore files that are older (having a less recent file modification time) than a pre-existing file or archive member with the same name. If the <code>-r</code> option is specified and the <code>-w</code> option is not specified, an archive member with the same name as a file in the file system shall be extracted if the archive member is newer than the file. If the <code>-w</code> option is specified and the <code>-r</code> option is not specified, an archive file member with the same name as a file in the file system shall be superseded if the file is newer than the archive member. It is unspecified if this is accomplished by actual replacement in the archive or by appending to the archive. If both the <code>-r</code> and <code>-w</code> options are specified, the file in the destination hierarchy shall be replaced by the file in the source hierarchy or by a link to the file in the source hierarchy if the file in the source hierarchy is newer.
10022		
10023		
10024		
10025		
10026		
10027		
10028		
10029		
10030		
10031		
10032		
10033		
10034		
10035	<code>-v</code>	Produce a verbose table of contents (see 4.48.6.1) if neither the <code>-r</code> option nor the <code>-w</code> option is specified. Otherwise, list archive member pathnames to standard error (see 4.48.6.2).
10036		
10037		
10038	<code>-x format</code>	Specify the output archive format. The <code>pax</code> utility shall recognize the following formats:
10039		
10040	<code>cpio</code>	The extended <code>cpio</code> interchange format specified in
10041		POSIX.1 {8} 10.1.2. The default <i>blocksize</i> for this format for character special archive files shall be 5120.
10042		1
10043		Implementations shall support all <i>blocksize</i> values
10044		less than or equal to 32 256 that are multiples of
10045		512.
10046	<code>ustar</code>	The extended <code>tar</code> interchange format specified in
10047		POSIX.1 {8} 10.1.1. The default <i>blocksize</i> for this format for character special archive files shall be
10048		10 240. Implementations shall support all <i>blocksize</i>
10049		1

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

10050 values less than or equal to 32 256 that are multiples 1
 10051 of 512.

10052 Implementation-defined formats shall specify a default block size
 10053 as well as any other block sizes supported for character special
 10054 archive files.

10055 Any attempt to append to an archive file in a format different
 10056 from the existing archive format shall cause `pax` to exit immedi-
 10057 ately with a nonzero exit status.

10058 `-X` When traversing the file hierarchy specified by a pathname, `pax`
 10059 shall not descend into directories that have a different device ID
 10060 [`st_dev`, see POSIX.1 {8} `stat()`].

10061 The options that operate on the names of files or archive members (`-c`, `-i`, `-n`, `-s`, 1
 10062 `-u`, and `-v`) shall interact as follows. When the `-r` option is specified and the `-w` 1
 10063 option is not (archive members are being extracted), the archive members shall be 1
 10064 “selected,” based on the user-specified *pattern* operands as modified by the `-c`, `-n`,
 10065 and `-u` options. Then, any `-s` and `-i` options shall modify, in that order, the
 10066 names of the selected files. The `-v` option shall write names resulting from these
 10067 modifications.

10068 When the `-w` option is specified (files are being archived), the files shall be
 10069 selected based on the user-specified pathnames as modified by the `-n` and `-u`
 10070 options. Then, any `-s` and `-i` options shall, in that order, modify the names of
 10071 these selected files. The `-v` option shall write names resulting from these
 10072 modifications. 1

10073 If both the `-u` and `-n` options are specified, `pax` shall not consider a file selected
 10074 unless it is newer than the file to which it is compared.

10075 4.48.4 Operands

10076 The following operands shall be supported by the implementation:

10077	<i>directory</i>	The destination directory pathname for copies when both the <code>-r</code> 10078 and <code>-w</code> options are specified.	
10079	<i>file</i>	A pathname of a file to be copied or archived.	
10080	<i>pattern</i>	A pattern matching one or more pathnames of archive members. 10081 A pattern shall be given in the name-generating notation of the 10082 pattern matching notation in 3.13, including the filename expansion 10083 rules in 3.13.3. The default, if no <i>pattern</i> is specified, is to 1 10084 select all members in the archive.	

10085 4.48.5 External Influences

10086 4.48.5.1 Standard Input

10087 If the `-w` option is specified, the standard input shall be used only if no *file*
 10088 operands are specified. It shall be a text file containing a list of pathnames, one
 10089 per line, without leading or trailing `<blank>`s.

10090 If neither the `-f` nor `-w` options are specified, the standard input shall be an
 10091 archive file. (See 4.48.5.2.)

10092 Otherwise, the standard input shall not be used.

10093 4.48.5.2 Input Files

10094 The input file named by the *archive* option-argument, or standard input when the
 10095 archive is read from there, shall be a file formatted according to one of the
 10096 specifications in POSIX.1 {8} 10.1, or some other, implementation-defined, format.

10097 The file `/dev/tty` shall be used to write prompts and read responses.

10098 4.48.5.3 Environment Variables

10099 The following environment variables shall affect the execution of `pax`:

10100	LANG	This variable shall determine the locale to use for the
10101		locale categories when both LC_ALL and the correspond-
10102		ing environment variable (beginning with LC_) do not
10103		specify a locale. See 2.6.
10104	LC_ALL	This variable shall determine the locale to be used to over-
10105		ride any values for locale categories specified by the set-
10106		tings of LANG or any environment variables beginning
10107		with LC_ .
10108	LC_COLLATE	This variable shall determine the locale for the behavior of
10109		ranges, equivalence classes, and multicharacter collating
10110		elements used in the pattern matching expressions for the
10111		<i>pattern</i> operand, the basic regular expression for the <code>-s</code>
10112		option, and the extended regular expression defined for
10113		the <code>yesexpr</code> locale keyword in the LC_MESSAGES
10114		category.
10115	LC_CTYPE	This variable shall determine the locale for the interpreta-
10116		tion of sequences of bytes of text data as characters (e.g.,
10117		single- versus multibyte characters in arguments and
10118		input files) and the behavior of character classes within
10119		regular expressions and pattern matching.

10120 **LC_MESSAGES** This variable shall determine the processing of affirmative
 10121 responses and the language in which messages should be
 10122 written.

10123 **LC_TIME** This variable shall determine the format and contents of
 10124 date and time strings when the `-v` option is specified.

10125 **4.48.5.4 Asynchronous Events**

10126 Default.

10127 **4.48.6 External Effects**

10128 **4.48.6.1 Standard Output**

10129 If the `-w` option is specified and neither the `-f` nor `-r` options are specified, the
 10130 standard output shall be the archive formatted according to one of the
 10131 specifications in POSIX.1 {8} 10.1, or some other implementation-defined format.
 10132 (See `-x format` under 4.48.3.)

10133 If neither the `-r` option nor the `-w` option is specified, the table of contents of the
 10134 selected archive members shall be written to standard output using the following
 10135 format:

10136 "`%s\n`", *<pathname>*

10137 If neither the `-r` option nor the `-w` option is specified, but the `-v` option is
 10138 specified, the table of contents of the selected archive members shall be written to
 10139 standard output using the following formats:

10140 For pathnames representing hard links to previous members of the archive:

10141 "`%sΔ=Δ%s\n`", *<ls -l listing>*, *<linkname>*

10142 For all other pathnames:

10143 "`%s\n`", *<ls -l listing>*

10144 where *<ls -l listing>* shall be the format specified by the `ls` utility (see 4.39) with
 10145 the `-l` option. When writing pathnames in this format, it is unspecified what is
 10146 written for fields for which the underlying archive format does not have the
 10147 correct information, although the correct number of `<blank>`-separated fields
 10148 shall be written.

10149 When writing a table of contents of selected archive members, standard output
 10150 shall not be buffered more than a line at a time.

10151 **4.48.6.2 Standard Error**

10152 If either or both of the `-r` option and the `-w` option are specified as well as the `-v`
 10153 option, `pax` shall write the pathnames it processes to the standard error output
 10154 using the following format:

10155 "%s\n", <pathname>

10156 These pathnames shall be written as soon as processing is begun on the file or
 10157 archive member, and shall be flushed to standard error. The trailing <newline>,
 10158 which shall not be buffered, shall be written when the file has been read or
 10159 written.

10160 If the `-s` option is specified, and the replacement string has a trailing `p`, substitu-
 10161 tions shall be written to standard error in the following format:

10162 "%sΔ>Δ%s\n", <original pathname>, <new pathname>

2

10163 In all operating modes of `pax` (see 4.48.2), optional messages of unspecified format
 10164 concerning the input archive format and volume number, the number of files,
 10165 blocks, volumes, and media parts as well as other diagnostic messages may be
 10166 written to standard error.

10167 In all formats, for both standard output and standard error, it is unspecified how
 10168 nonprintable characters in pathnames or linknames are written.

10169 **4.48.6.3 Output Files**

10170 If the `-r` option is specified, the extracted or copied output files shall be of the
 10171 archived file type.

10172 If the `-w` option is specified, but the `-r` option is not, the output file named by the
 10173 `-f` option argument shall be a file formatted according to one of the specifications
 10174 in POSIX.1 {8} 10.1, or some other, implementation-defined, format.

10175 **4.48.7 Extended Description**

10176 None.

10177 **4.48.8 Exit Status**

10178 The `pax` utility shall exit with one of the following values:

10179 0 All files were processed successfully.

10180 >0 An error occurred.

10181 **4.48.9 Consequences of Errors**

10182 If `pax` cannot create a file or a link when reading an archive or cannot find a file
 10183 when writing an archive, or cannot preserve the user ID, group ID, or file mode
 10184 when the `-p` option is specified, a diagnostic message shall be written to standard
 10185 error and a nonzero exit status shall be returned, but processing shall continue.
 10186 In the case where `pax` cannot create a link to a file, `pax` shall not, by default,
 10187 create a second copy of the file.

10188 If the extraction of a file from an archive is prematurely terminated by a signal or
 10189 error, `pax` may have only partially extracted the file or (if the `-n` option was not
 10190 specified) may have extracted a file of the same name as that specified by the
 10191 user, but which is not the file the user wanted. Additionally, the file modes of
 10192 extracted directories may have additional bits from the `S_IRWXU` mask set as well
 10193 as incorrect modification and access times.

10194 **4.48.10 Rationale.** *(This subclause is not a part of P1003.2)*

10195 **Examples, Usage**

10196 The following command:

```
10197     pax -w -f /dev/rmt/lm .
```

10198 copies the contents of the current directory to tape drive 1, medium density
 10199 (assuming historical System V device naming procedures. The historical BSD dev-
 10200 ice name would be `/dev/rmt9`).

10201 The following commands:

```
10202     mkdir newdir
10203     pax -rw olddir newdir
```

10204 copy the `olddir` directory hierarchy to `newdir`.

```
10205     pax -r -s ',^//*usr/*,, ' -f a.pax
```

10206 reads the archive `a.pax`, with all files rooted in `"/usr"` in the archive extracted
 10207 relative to the current directory.

10208 The `-p` (privileges) option was invented to reconcile differences between historical 1
 10209 `tar` and `cpio` implementations. In particular, the two utilities used `-m` in 1
 10210 diametrically opposed ways. The `-p` option also provides a consistent means of 1
 10211 extending the ways in which future file attributes can be addressed, such as for 1
 10212 enhanced security systems or high-performance files. Although it may seem com- 1
 10213 plex, there are really two modes that will be most commonly used: 1

10214	<code>-p e</code>	“Preserve everything.” This would be used by the historical super- 1	1
10215		user, someone with all the appropriate privileges, to preserve all 1	1
10216		aspects of the files as they are recorded in the archive. The <code>e</code> flag is 1	1
10217		the sum of <code>o</code> and <code>p</code> , and other implementation-defined attributes. 1	1
10218	<code>-p p</code>	“Preserve” the file mode bits. This would be used by the user with 1	1
10219		regular privileges who wished to preserve aspects of the file other 1	1
10220		than the ownership. The file times are preserved by default, but two 1	1
10221		other flags are offered to disable these and use the time of extraction. 1	1

10222 **History of Decisions Made**

10223 The description of `pax` was adopted from a command written by Glenn Fowler of
 10224 AT&T. It is a new utility, commissioned for this standard.

10225 The table of contents output is written to standard output to facilitate pipeline
 10226 processing.

10227 The output archive formats required are those defined in POSIX.1 {8}; others, such
 10228 as the historical `tar` format, may be added as an extension.

10229 The one pathname per line format of standard input precludes pathnames con-
 10230 taining `<newline>`s. Although such pathnames violate the portable filename
 10231 guidelines, they may exist and their presence may inhibit usage of `pax` within
 10232 shell scripts. This problem is inherited from historical archive programs. The
 10233 problem can be avoided by listing filename arguments on the command line
 10234 instead of on standard input.

10235 An earlier draft had hard links displaying for all pathnames. This was removed 1
 10236 because it complicates the output of the non `-v` case and does not match historical 1
 10237 `cpio` usage. The hard-link information is available in the `-v` display. 1

10238 The working group realizes that the presence of symbolic links will affect certain
 10239 `pax` operations. Historical practice, in both System V and BSD-based systems, is
 10240 that the physical traversal of the file hierarchy shall be the default, and an option
 10241 is provided to cause the utility to do a logical traversal, that is, follow symbolic
 10242 links. Historical practice has not been so consistent as to what option is used to
 10243 cause the logical traversal; BSD systems have used `-h` (`cp` and `tar`) and `-L` (`ls`),
 10244 while the *SVID* specifies `-L` (`cpio` and `ls`). Given this inconsistency, the `-L`
 10245 option is recommended.

10246 The archive formats described in POSIX.1 {8} have certain restrictions that have
 10247 been brought along from historical usage. For example, there are restrictions on
 10248 the length of pathnames stored in the archive. When `pax` is used in `-rw` mode,
 10249 copying directory hierarchies, there is no stated dependency on these archive for-
 10250 mats. Therefore, such restrictions should not apply.

10251 The POSIX.2 working group is currently devising a new archive format to be pub- 1
 10252 lished in a revision or amendment to this standard. It is expected that the `ustar` 1
 10253 and `cpio` formats then will be retired from a future version of POSIX.1 {8}. This 1
 10254 new format will address all restrictions and new requirements for security label-
 10255 ing, etc. The `pax` utility should be upward-compatible enough to handle any such
 10256 changes. The reason that the default `-x format` output format is implementation
 10257 defined is to reserve the default format for this new standard interface. The `-o` 1
 10258 option was devised to provide means of controlling the many aspects of interna- 1
 10259 tional and security concerns without expending the entire alphabet of option 1
 10260 letters for this, and possibly other, file formats. The `-o` string is meant to be 1
 10261 specific for each `-x` format. Control of various file permissions and attributes that 1
 10262 can be expressed in a binary way will continue to use the `-p` (permissions) option; 1
 10263 the `-o` will be reserved for more involved requirements and will probably take a 1

10264 `pax -o name=value,name=value -o name=value` 1

10265 approach. 1

10266 The fundamental difference in how `cpio` and `tar` viewed the world was in the
 10267 way directories were treated. The `cpio` utility did not treat directories differently
 10268 from other files, and to select a directory and its contents required that each file

10269 in the hierarchy be explicitly specified. For `tar`, a directory matched every file in
10270 the file hierarchy it rooted.

10271 The `pax` utility offers both interfaces; by default, directories map into the file
10272 hierarchy they root. The `-d` option causes `pax` to skip any file not explicitly refer-
10273 enced, as `cpio` traditionally did. The `tar`-style behavior was chosen as the
10274 default because it was believed that this was the more common usage, and
10275 because `tar` is the more commonly available interface, as it was historically pro-
10276 vided on both System V and BSD implementations. Because a file may be
10277 matched more than once without causing it to be selected multiple times, the
10278 traditional usage of piping an `ls` or `find` to the archive command works as
10279 always.

10280 The Data Interchange Format specification of POSIX.1 {8} requires that processes
10281 with “appropriate privileges” shall always restore the ownership and permissions
10282 of extracted files exactly as archived. If viewed from the historic equivalence
10283 between super-user and “appropriate privileges,” there are two problems with
10284 this requirement. First, users running as super-users may unknowingly set
10285 dangerous permissions on extracted files. Second, it is needlessly limiting in that
10286 super-users cannot extract files and own them as super-user unless the archive
10287 was created by the super-user. (It should be noted that restoration of ownerships
10288 and permissions for the super-user, by default, is historical practice in `cpio`, but
10289 not in `tar`.) In order to avoid these two problems, the `pax` specification has an
10290 additional “privilege” mechanism, the `-p` option. Only a `pax` invocation with the
10291 POSIX.1 {8} privileges needed, and which has the `-p` option set using the `e`
10292 specification character, has the “appropriate privilege” to restore full ownership
10293 and permission information.

10294 Note also that POSIX.1 {8} 10.1 requires that the file ownership and access per-
10295 missions shall be set, on extraction, in the same fashion as the POSIX.1 {8} `creat()`
10296 function when provided the mode stored in the archive. This means that the file
10297 creation mask of the user is applied to the file permissions.

10298 The default `blocksize` value of 5120 for `cpio` was selected because it is one of the
10299 standard block-size values for `cpio`, set when the `-B` option is specified. (The
10300 other default block-size value for `cpio` is 512, and this was felt to be too small.)
10301 The default block value of 10240 for `tar` was selected as that is the standard
10302 block-size value for BSD `tar`. The maximum block size of 32 256 (2^{15} –512) is the
10303 largest multiple of 512 that fits into a signed 16-bit tape controller transfer regis-
10304 ter. There are known limitations in some historic system that would prevent
10305 larger blocks from being accepted. Historic values were chosen to make compati-
10306 bility with existing scripts using `dd` or similar utilities to manipulate archives
10307 more likely. Also, default block sizes for any file type other than character special
10308 has been deleted from the standard as unimportant and not likely to affect the
10309 structure of the resulting archive.

10310 Implementations are permitted to modify the block-size value based on the
10311 archive format or the device to which the archive is being written. This is to pro-
10312 vide implementations the opportunity to take advantage of special types of dev-
10313 ices, and should not be used without a great deal of consideration as it will almost
10314 certainly decrease archive portability.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

The `-n` option in early drafts had three effects; the first was to cause special characters in patterns to not be treated specially. The second was to cause only the first file that matched a pattern to be extracted. The third was to cause `pax` to write a diagnostic message to standard error when no file was found matching a specified pattern. Only the second behavior is retained by POSIX.2, for many reasons. First, it is in general a bad idea for a single option to have multiple effects. Second, the ability to make pattern matching characters act as normal characters is useful for other parts of `pax` than just file extraction. Third, a finer degree of control over the special characters is useful, because users may wish to normalize only a single special character in a single file name. Fourth, given a more general escape mechanism, the previous behavior of the `-n` option can be easily obtained using the `-s` option or a `sed` script. Finally, writing a diagnostic message when a pattern specified by the user is unmatched by any file is useful behavior in all cases.

There are two methods of copying subtrees in POSIX.2. The other method is described as part of the `cp` utility (see 4.13). Both methods are historical practice: `cp` provides a simpler, more intuitive interface, while `pax` offers a finer granularity of control. Each provides additional functionality to the other; in particular, `pax` maintains the hard-link structure of the hierarchy, while `cp` does not. It is the intention of the working group that the results be similar (using appropriate option combinations in both utilities). The results are not required to be identical; there seemed insufficient gain to applications to balance the difficulty of implementations having to guarantee that the results would be exactly identical.

A single archive may span more than one file. See POSIX.1 [8] 10.1.3. While POSIX.1 [8] only refers to reading the archive file, it is reasonable that the format utility may also determine, in an implementation-defined manner, the next file to write. It is suggested that implementations provide informative messages to the user on the standard error whenever the archive file is changed.

The `-d` option (do not create intermediate directories not listed in the archive) found in previous drafts of this standard was originally provided as a complement to the historic `-d` option of `cpio`. It has been deleted.

The `-s` option in earlier drafts specified a subset of the substitution command from the `ed` utility. As there was no reason for only a subset to be supported, the `-s` option is now compatible with the current `ed` specification. Since the delimiter can be any nonnull character, the following usage with single spaces is valid:

```
pax -s " foo bar " ...
```

The `-t` option (specify an implementation-defined identifier naming an input or output device) found in earlier drafts has been deleted because it is not historical practice and of limited utility. In particular, historic versions of neither `cpio` nor `tar` had the concept of devices that were not mapped into the file system; if the devices are mapped into the file system, the `-f` option is sufficient.

The `-o` and `-p` options found in previous versions of this standard have been renamed to be `-p` and `-t`, respectively, to correspond more closely with the historic `tar` and `cp` utilities.

10359 The default behavior of `pax` with regard to file modification times is the same as
10360 historical implementations of `tar`. It is not the historical behavior of `cpio`.

10361 Because the `-i` option uses `/dev/tty`, utilities without a controlling terminal will
10362 not be able to use this option.

10363 The `-y` option, found in earlier drafts, has been deleted because a line containing
10364 a single period for the `-i` option has equivalent functionality. The special lines
10365 for the `-i` option (a single period and the empty line) are historical practice in
10366 `cpio`.

10367 In earlier drafts, an `-e charmap` option was included to increase portability of 1
10368 files between systems using different coded character sets. This option was omit- 1
10369 ted because it was apparent that consensus could not be formed for it. It was an 1
10370 interface without implementation experience and overloaded the `charmap` file 1
10371 concept to provide additional uses its original authors had not intended. The 1
10372 developers of POSIX.2 will consider other mechanisms for transporting files with 1
10373 nonportable names as they develop the new interchange format, described earlier. 1

10374 The `-k` option was added to address international concerns about the dangers
10375 involved in the character set transformations of `-e` (if the target character set
10376 were different than the source, the file names might be transformed into names
10377 matching existing files) and was made more general to also protect files
10378 transferred between file systems with different `{NAME_MAX}` values (truncating a
10379 filename on a smaller system might also inadvertently overwrite existing files).
10380 As stated, it prevents any overwriting, even if the target file is older than the
10381 source, which is seen as a generally useful feature anyway.

10382 It is almost certain that appropriate privileges will be required for `pax` to accom-
10383 plish parts of this specification. Specifically, creating files of type block special or
10384 character special, restoring file access times unless the files are owned by the user
10385 (the `-t` option), or preserving file owner, group, and mode (the `-p` option) will all
10386 probably require appropriate privileges.

10387 Some of the file characteristics referenced in this specification may not be sup-
10388 ported by some archive formats. For example, neither the `tar` nor `cpio` formats
10389 contain the file access time. For this reason, the `e` specification character has
10390 been provided, intended to cause all file characteristics specified in the archive to
10391 be retained.

10392 It is required that extracted directories, by default, have their access and
10393 modification times and permissions set to the values specified in the archive.
10394 This has obvious problems in that the directories are almost certainly modified
10395 after being extracted and that directory permissions may not permit file creation.
10396 One possible solution is to create directories with the mode specified in the
10397 archive, as modified by the `umask` of the user, plus sufficient permissions to allow
10398 file creation. After all files have been extracted, `pax` would then reset the access
10399 and modification times and permissions as necessary.

10400 When the `-r` option is specified, and the `-w` option is not, implementations are
10401 permitted to overwrite files when the archive has multiple members with the
10402 same name. This may fail, of course, if permissions on the first version of the file

do not permit it to be overwritten.

4.49 pr — Print files

4.49.1 Synopsis

```
pr [+page] [-column] [-adFmrt] [-e[char][gap]] [-h header] [-i[char][gap]]
    [-l lines] [-n[char][width]] [-o offset] [-s[char]] [-w width] [file ...]
```

4.49.2 Description

The `pr` utility is a printing and pagination filter. If multiple input files are specified, each shall be read, formatted, and written to standard output. By default, the input shall be separated into 66-line pages, each with:

- A 5-line header that includes the page number, date, time, and the path-name of the file. 1

- A 5-line trailer consisting of blank lines. 1

If standard output is associated with a terminal, diagnostic messages shall be deferred until the `pr` utility has completed processing.

When options specifying multicolumn output are specified, output text columns shall be of equal width; input lines that do not fit into a text column shall be truncated. By default, text columns shall be separated with at least one <blank>.

4.49.3 Options

The `pr` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that: the *page* option has a '+' delimiter; *page* and *column* can be multidigit numbers; some of the option-arguments are optional; and some of the option-arguments cannot be specified as separate arguments from the preceding option letter. In particular, the `-s` option does not allow the option letter to be separated from its argument, and the options `-e`, `-i`, and `-n` require that both arguments, if present, not be separated from the option letter.

The following options shall be supported by the implementation. In the following option descriptions, *column*, *lines*, *offset*, *page*, and *width* are positive decimal integers; *gap* is a nonnegative decimal integer. 1

10431	<code>+page</code>	Begin output at page number <i>page</i> of the formatted input.	1
10432	<code>-column</code>	Produce output that is <i>columns</i> wide (default shall be 1) and is written down each column in the order in which the text is received from the input file. This option should not be used with <code>-m</code> . The options <code>-e</code> and <code>-i</code> shall be assumed for multiple text-column output. Whether or not text columns are balanced is	1

10437		unspecified, but a text column shall never exceed the length of the	
10438		page (see the <code>-l</code> option). When used with <code>-t</code> , use the minimum	
10439		number of lines to write the output.	
10440	<code>-a</code>	Modify the effect of the <code>-column</code> option so that the columns are	1
10441		filled across the page in a round-robin order (e.g., when <code>column</code> is	1
10442		2, the first input line heads column 1, the second heads column 2,	1
10443		the third is the second line in column 1, etc.).	1
10444	<code>-d</code>	Produce output that is double-spaced; append an extra <code><new-</code>	
10445		<code>line></code> following every <code><newline></code> found in the input.	
10446	<code>-e[<i>char</i>][<i>gap</i>]</code>		
10447		Expand each input <code><tab></code> to the next greater column position	1
10448		specified by the formula $n*gap+1$, where n is an integer > 0 . If	1
10449		<code>gap</code> is zero or is omitted, it shall default to 8. All <code><tab></code> charac-	
10450		ters in the input shall be expanded into the appropriate number	
10451		of <code><space></code> s. If any nondigit character, <code>char</code> , is specified, it shall	
10452		be used as the input tab character.	
10453	<code>-F</code>	Use a <code><form-feed></code> character for new pages, instead of the	
10454		default behavior that uses a sequence of <code><newline></code> characters.	
10455	<code>-h header</code>	Use the string <code>header</code> to replace the contents of the <code>file</code> operand in	1
10456		the page header. See 4.49.6.1.	1
10457	<code>-i[<i>char</i>][<i>gap</i>]</code>		
10458		In output, replace multiple <code><space></code> s with <code><tab></code> s wherever two	
10459		or more adjacent <code><space></code> s reach column positions <code>gap+1</code> ,	
10460		$2*gap+1$, $3*gap+1$, etc. If <code>gap</code> is zero or is omitted, default <code><tab></code>	
10461		settings at every eighth column position shall be assumed. If any	
10462		nondigit character, <code>char</code> , is specified, it shall be used as the out-	
10463		put <code><tab></code> character.	
10464	<code>-l lines</code>	Override the 66-line default and reset the page length to <code>lines</code> . If	
10465		<code>lines</code> is not greater than the sum of both the header and trailer	1
10466		depths (in lines), the <code>pr</code> utility shall suppress both the header	
10467		and trailer, as if the <code>-t</code> option were in effect.	
10468	<code>-m</code>	Merge files. Standard output shall be formatted so the <code>pr</code> utility	
10469		writes one line from each file specified by a <code>file</code> operand, side by	
10470		side into text columns of equal fixed widths, in terms of the	
10471		number of column positions. Implementations shall support	
10472		merging of at least nine <code>file</code> operands.	
10473	<code>-n[<i>char</i>][<i>width</i>]</code>		
10474		Provide <code>width</code> -digit line numbering (default for <code>width</code> shall be 5).	
10475		The number shall occupy the first <code>width</code> column positions of each	1
10476		text column of default output or each line of <code>-m</code> output. If <code>char</code>	
10477		(any nondigit character) is given, it shall be appended to the line	
10478		number to separate it from whatever follows (default for <code>char</code>	
10479		shall be a <code><tab></code>).	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

- 10480 `-o offset` Each line of output shall be preceded by offset `<space>s`. If the `-o`
 10481 option is not specified, the default offset shall be zero. The space
 10482 taken shall be in addition to the output line width (see `-w` option
 10483 below).
- 10484 `-r` Write no diagnostic reports on failure to open files.
- 10485 `-s[char]` Separate text columns by the single character *char* instead of by
 10486 the appropriate number of `<space>s` (default for *char* shall be
 10487 the `<tab>` character).
- 10488 `-t` Write neither the five-line identifying header nor the five-line
 10489 trailer usually supplied for each page. Quit writing after the last
 10490 line of each file without spacing to the end of the page.
- 10491 `-w width` Set the width of the line to *width* column positions for multiple
 10492 text-column output only. If the `-w` option is not specified and the
 10493 `-s` option is not specified, the default width shall be 72. If the `-w`
 10494 option is not specified and the `-s` option is specified, the default
 10495 width shall be 512.
- 10496 For single column output, input lines shall not be truncated.

10497 **4.49.4 Operands**

10498 The following operand shall be supported by the implementation:

- 10499 *file* A pathname of a file to be written. If no *file* operands are
 10500 specified, or if a *file* operand is `-`, the standard input shall be
 10501 used.

10502 **4.49.5 External Influences**

10503 **4.49.5.1 Standard Input**

10504 The standard input shall be used only if no *file* operands are specified, or if a *file*
 10505 operand is `-`. See Input Files.

10506 **4.49.5.2 Input Files**

10507 The input files shall be text files.

10508 **4.49.5.3 Environment Variables**

10509 The following environment variables shall affect the execution of `pr`:

- 10510 **LANG** This variable shall determine the locale to use for the
 10511 locale categories when both `LC_ALL` and the correspond-
 10512 ing environment variable (beginning with `LC_`) do not
 10513 specify a locale. See 2.6.

10514	LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
10515		
10516		
10517		
10518	LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files) and which characters are defined as printable (character class <code>print</code>). Nonprintable characters still shall be written to standard output, but shall be not counted for the purpose for column-width and line-length calculations.
10519		
10520		
10521		
10522		
10523		
10524		
10525		
10526	LC_MESSAGES	This variable shall determine the language in which messages should be written.
10527		
10528	LC_TIME	This variable shall determine the format of the date and time for use in writing header lines.
10529		
10530	TZ	This variable shall determine the time zone for use in writing header lines.
10531		

10532 **4.49.5.4 Asynchronous Events**

10533 If `pr` receives an interrupt while writing to a terminal, it shall flush all accumulated error messages to the screen before terminating.

10535 **4.49.6 External Effects**

10536 **4.49.6.1 Standard Output**

10537 The `pr` utility output shall be a paginated version of the original file (or files).
 10538 This pagination shall be accomplished using either `<form-feed>s` or a sequence
 10539 of `<newline>s`, as controlled by the `-F` option. Page headers shall be generated
 10540 unless the `-t` option is specified. The page headers shall be of the form:

```
10541     "\n\n%s %s Page %d\n\n\n", <output of date>, <file>,
10542     <page number>
```

10543 In the POSIX Locale, the `<output of date>` field, representing the date and time of
 10544 last modification of the input file (or the current date and time if the input file is
 10545 standard input), shall be equivalent to the output of the following command as it
 10546 would appear if executed at the given time:

```
10547     date "+%b %e %H:%M %Y"
```

10548 without the trailing `<newline>`, if the page being written is from standard input.
 10549 If the page being written is not from standard input, in the POSIX Locale, the
 10550 same format shall be used, but the time used shall be the modification time of the
 10551 file corresponding to `file` instead of the current time. When the `LC_TIME` locale

category is not set to the POSIX Locale, a different format and order of presentation of this field may be used.

If the standard input is used instead of a *file* operand, the *<file>* field shall be replaced by a null string.

If the *-h* option is specified, the *file* field shall be replaced by the *header* argument.

10558 **4.49.6.2 Standard Error**

Used only for diagnostic messages.

10560 **4.49.6.3 Output Files**

None.

10562 **4.49.7 Extended Description**

None.

10564 **4.49.8 Exit Status**

The *pr* utility shall exit with one of the following values:

0 All files were written successfully.

>0 An error occurred.

10568 **4.49.9 Consequences of Errors**

Default.

10570 **4.49.10 Rationale.** (*This subclause is not a part of P1003.2*)

10571 **Examples, Usage**

To print a numbered list of all files in the current directory:

```
10573     ls -a | pr -n -h "Files in $(pwd)."
```

10574 **History of Decisions Made**

This utility is one of those that does not follow the Utility Syntax Guidelines because of its historical origins. The working group could have added new options that obeyed the guidelines (and marked the old options *obsolescent*) or devised an entirely new utility; there are examples of both actions in this standard. For this utility, it chose to leave some of the options as they are because of their heavy usage by existing applications. However, due to interest in the international

1

community, the developers of the standard have agreed to provide an alternative syntax for the next version of this standard that conforms to the spirit of the Utility Syntax Guidelines. This new syntax will be accompanied by the existing syntax, marked as obsolescent. System implementors are encouraged to develop and promulgate a new syntax for `pr`, perhaps using a different utility name, that can be adopted for the next version of this standard.

Implementations are required to accept option arguments to the `-h`, `-l`, `-o`, and `-w` options whether presented as part of the same argument or as a separate argument to `pr`, as suggested by the utility syntax guidelines. The `-n` and `-s` options, however, are specified as in historical practice because they are frequently specified without their optional arguments. If a `<blank>` were allowed before the option-argument in these cases, a file operand could mistakenly be interpreted as an option-argument in historical applications.

Historical implementations of the `pr` utility have differed in the action taken for the `-f` option. BSD uses it as described here for the `-F` option; System V uses it to change trailing `<newline>`s on each page to a `<form-feed>` and, if standard output is a TTY device, sends an `<alert>` to standard error and reads a line from `/dev/tty` before the first page. Draft 9 incorrectly specified part of the System V behavior, raising several ballot objections. There were strong arguments from both sides of this issue concerning existing practice and additional arguments against the System V `-f` behavior, on the grounds that it was not a modular design to have the behavior of an option change depending on where output is directed. Therefore, the `-f` option is not specified and the `-F` option has been added.

The `-p` option was omitted since it represents a purely interactive usage.

1

The `<output of date>` field in the `-l` format is specified only for the POSIX Locale. As noted, the format can be different in other locales. No mechanism for defining this is present in this standard, as the appropriate vehicle is a messaging system; i.e., the format should be specified as a “message.”

10610 **4.50 printf — Write formatted output**

10611 **4.50.1 Synopsis**

10612 `printf` *format* [*argument* ...]

10613 **4.50.2 Description**

10614 The `printf` utility shall write formatted operands to the standard output. The
10615 *argument* operands shall be formatted under control of the *format* operand.

10616 **4.50.3 Options**

10617 None.

10618 **4.50.4 Operands**

10619 The following operands shall be supported by the implementation:

10620 *format* A string describing the format to use to write the remaining
10621 operands; see 4.50.7.

10622 *argument* The strings to be written to standard output, under the control of
10623 *format*; see 4.50.7.

10624 **4.50.5 External Influences**

10625 **4.50.5.1 Standard Input**

10626 None.

10627 **4.50.5.2 Input Files**

10628 None.

10629 **4.50.5.3 Environment Variables**

10630 The following environment variables shall affect the execution of `printf`:

10631 **LANG** This variable shall determine the locale to use for the
10632 locale categories when both **LC_ALL** and the correspond-
10633 ing environment variable (beginning with **LC_**) do not
10634 specify a locale. See 2.6.

10635	LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
10636		
10637		
10638		
10639	LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).
10640		
10641		
10642	LC_MESSAGES	This variable shall determine the language in which messages should be written.
10643		
10644	LC_NUMERIC	This variable shall determine the locale for numeric formatting. It shall affect the format of numbers written using the e , E , f , g , and G conversion characters (if supported).
10645		
10646		
10647		

10648 **4.50.5.4 Asynchronous Events**

10649 Default.

10650 **4.50.6 External Effects**

10651 **4.50.6.1 Standard Output**

10652 See 4.50.7.

10653 **4.50.6.2 Standard Error**

10654 Used only for diagnostic messages.

10655 **4.50.6.3 Output Files**

10656 None.

10657 **4.50.7 Extended Description**

10658 The *format* operand shall be used as the *format* string described in 2.12 with the
10659 following exceptions:

- 10660 (1) A <space> character in the format string, in any context other than a
10661 flag of a conversion specification, shall be treated as an ordinary character
10662 that is copied to the output.
- 10663 (2) A Δ character in the format string shall be treated as a Δ character, not as
10664 a <space>.
- 10665 (3) In addition to the escape sequences shown in Table 2-15 (see 2.12), *\ddd*,
10666 where *ddd* is a one-, two-, or three-digit octal number, shall be written as
10667 a byte with the numeric value specified by the octal number.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

- 10668 (4) The implementation shall not precede or follow output from the `d` or `u`
 10669 conversion specifications with `<blank>`s not specified by the *format*
 10670 operand.
- 10671 (5) The implementation shall not precede output from the `o` conversion
 10672 specification with zeroes not specified by the *format* operand.
- 10673 (6) The `e`, `E`, `f`, `g`, and `G` conversion specifications need not be supported.
- 10674 (7) An additional conversion character, `b`, shall be supported as follows. The
 10675 argument shall be taken to be a string that may contain backslash-
 10676 escape sequences. The following backslash-escape sequences shall be
 10677 supported:
- 10678 (a) The escape sequences listed in Table 2-15, which shall be converted
 10679 to the characters they represent;
 - 10680 (b) `\0ddd`, where *ddd* is a zero-, one-, two-, or three-digit octal number
 10681 that shall be converted to a byte with the numeric value specified by
 10682 the octal number;
 - 10683 (c) `\c`, which shall not be written and shall cause `printf` to ignore any
 10684 remaining characters in the string operand containing it, any
 10685 remaining string operands, and any additional characters in the
 10686 *format* operand.
- 10687 The interpretation of a backslash followed by any other sequence of char-
 10688 acters is unspecified.
- 10689 Bytes from the converted string shall be written until the end of the
 10690 string or the number of bytes indicated by the precision specification is
 10691 reached. If the precision is omitted, it shall be taken to be infinite, so all
 10692 bytes up to the end of the converted string shall be written.
- 10693 (8) For each specification that consumes an argument, the next argument
 10694 operand shall be evaluated and converted to the appropriate type for the
 10695 conversion as specified below.
- 10696 (9) The *format* operand shall be reused as often as necessary to satisfy the
 10697 argument operands. Any extra `c` or `s` conversion specifications shall be
 10698 evaluated as if a null string argument were supplied; other extra conver-
 10699 sion specifications shall be evaluated as if a zero argument were sup-
 10700 plied. If the *format* operand contains no conversion specifications and
 10701 *argument* operands are present, the results are unspecified.
- 10702 (10) If a character sequence in the *format* operand begins with a `%` character,
 10703 but does not form a valid conversion specification, the behavior is
 10704 unspecified.
- 10705 The *argument* operands shall be treated as strings if the corresponding conversion
 10706 character is `b`, `c`, or `s`; otherwise, it shall be evaluated as a C constant, as
 10707 described by the C Standard [7], with the following extensions:

- 10708 — A leading plus or minus sign shall be allowed.
- 10709 — If the leading character is a single- or double-quote, the value shall be the
- 10710 numeric value in the underlying code set of the character following the
- 10711 single- or double-quote.
- 10712 If an argument operand cannot be completely converted into an internal value
- 10713 appropriate to the corresponding conversion specification, a diagnostic message
- 10714 shall be written to standard error and the utility shall not exit with a zero exit
- 10715 status, but shall continue processing any remaining operands and shall write the
- 10716 value accumulated at the time the error was detected to standard output.

10717 **4.50.8 Exit Status**

10718 The `printf` utility shall exit with one of the following values:

- 10719 0 Successful completion.
- 10720 >0 An error occurred.

10721 **4.50.9 Consequences of Errors**

10722 Default.

10723 **4.50.10 Rationale.** *(This subclause is not a part of P1003.2)*

10724 **Examples, Usage**

10725 To alert the user and then print and read a series of prompts:

```
10726     printf "\aPlease fill in the following: \nName: "
10727     read name
10728     printf "Phone number: "
10729     read phone
```

10730 To read out a list of right and wrong answers from a file, calculate the percentage
 10731 right, and print them out. The numbers are right-justified and separated by a
 10732 single <tab>. The percentage is written to one decimal place of accuracy.

```
10733     while read right wrong ; do
10734         percent=$(echo "scale=1;($right*100)/($right+$wrong)" | bc)
10735         printf "%2d right\t%2d wrong\t(%s%%)\n" \
10736             $right $wrong $percent
10737     done < database_file
```

10738 The command:

```
10739     printf "%5d%4d\n" 1 21 321 4321 54321
```

10740 produces:

```
10741         1  21
10742       3214321
10743     54321    0
```

10744 Note that the *format* operand is used three times to print all of the given strings
10745 and that a 0 was supplied by `printf` to satisfy the last `%4d` conversion
10746 specification.

10747 The `printf` utility is required to notify the user when conversion errors are
10748 detected while producing numeric output; thus, the following results would be
10749 expected on an implementation with 32-bit twos-complement integers when `%d` is
10750 specified as the *format* operand:

Argument	Standard Output	Diagnostic Output
5a	5	<code>printf: "5a" not completely converted</code>
9999999999	2147483647	<code>printf: "9999999999" arithmetic overflow</code>
-9999999999	-2147483648	<code>printf: "-9999999999" arithmetic overflow</code>
ABC	0	<code>printf: "ABC" expected numeric value</code>

10757 The diagnostic message format is not specified, but these examples convey the
10758 type of information that should be reported. Note that the value shown on stan-
10759 dard output is what would be expected as the return value from the
10760 C Standard {7} function `strtol()`. A similar correspondence exists between `%u` and
10761 `strtoul()` and `%e`, `%f`, and `%g` (if the implementation supports floating-point conver-
10762 sions) and `strtod()`.

10763 In a locale using ISO/IEC 646 {1} as the underlying code set, the command:

```
10764     printf "%d\n" 3 +3 -3 \'3 \'"+3 "'-3"
```

10765 produces:

```
10766     3    Numeric value of constant 3
10767     3    Numeric value of constant 3
10768    -3    Numeric value of constant -3
10769    51    Numeric value of the character "3" in ISO/IEC 646 {1} code set
10770    43    Numeric value of the character "+" in ISO/IEC 646 {1} code set
10771    45    Numeric value of the character "-" in ISO/IEC 646 {1} code set
```

10772 Note that in a locale with multibyte characters, the value of a character is
10773 intended to be the value of the equivalent of the *wchar_t* representation of the
10774 character as described in C Standard {7}.

10775 History of Decisions Made

10776 The `printf` utility was added to provide functionality that has historically been
10777 provided by `echo`. However, due to irreconcilable differences in the various ver-
10778 sions of `echo` extant, the version in this standard has few special features, leav-
10779 ing those to this new `printf` utility, which is based on one in the Ninth Edition
10780 at AT&T Bell Labs.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

10781 The Extended Description almost exactly matches the C Standard {7} *printf()*
10782 function, although it is described in terms of the file format notation in 2.12.

10783 The floating point formatting conversion specifications are not required because
10784 all arithmetic in the shell is integer arithmetic. The *awk* utility performs floating
10785 point calculations and provides its own *printf* function. The *bc* utility can per-
10786 form arbitrary-precision floating point arithmetic, but doesn't provide extensive
10787 formatting capabilities. (This *printf* utility cannot really be used to format *bc*
10788 output; it does not support arbitrary precision.) Implementations are encouraged
10789 to support the floating point conversions as an extension.

10790 Note that this *printf* utility, like the C Standard {7} *printf()* function on which it
10791 is based, makes no special provision for dealing with multibyte characters when
10792 using the *%c* conversion specification or when a precision is specified in a *%b* or *%s*
10793 conversion specification. Applications should be extremely cautious using either
10794 of these features when there are multibyte characters in the character set.

10795 Field widths and precisions cannot be specified as *'*'* since the *'*'* can be
10796 replaced directly in the *format* operand using shell variable substitution. Imple-
10797 mentations can also provide this feature as an extension if they so choose.

10798 Hexadecimal character constants as defined in the C Standard {7} are not recog-
10799 nized in the *format* operand because there is no consistent way to detect the end
10800 of the constant. Octal character constants are limited to, at most, three octal
10801 digits, but hexadecimal character constants are only terminated by a nonhex-digit
10802 character. In the C Standard {7}, the *##* concatenation operator can be used to
10803 terminate a constant and follow it with a hexadecimal character to be written. In
10804 the shell, concatenation occurs before the *printf* utility has a chance to parse the
10805 end of the hexadecimal constant.

10806 The *%b* conversion specification is not part of the C Standard {7}; it has been
10807 added here as a portable way to process backslash-escapes expanded in string
10808 operands as provided by the System V version of the *echo* utility. See also the
10809 rationale for *echo* for ways to use *printf* as a replacement for all of the tradi-
10810 tional versions of the *echo* utility.

10811 If an argument cannot be parsed correctly for the corresponding conversion
10812 specification, the *printf* utility is required to report an error. Thus, overflow
10813 and extraneous characters at the end of an argument being used for a numeric
10814 conversion are to be reported as errors. If written in C, the *printf* utility could
10815 use the *strtol()* function to parse optionally signed numeric arguments, *strtoul()* to
10816 parse unsigned numeric arguments, and *strtod()* to parse floating point argu-
10817 ments (if floating point conversions are supported). It is not considered an error if
10818 an argument operand is not completely used for a *c* or *s* conversion or if a "string"
10819 operand's first or second character is used to get the numeric value of a character.

10820 **4.51 pwd — Return working directory name**

10821 **4.51.1 Synopsis**

10822 pwd

10823 **4.51.2 Description**

10824 The pwd utility shall write an absolute pathname of the current working directory
10825 to standard output.

10826 **4.51.3 Options**

10827 None.

10828 **4.51.4 Operands**

10829 None.

10830 **4.51.5 External Influences**

10831 **4.51.5.1 Standard Input**

10832 None.

10833 **4.51.5.2 Input Files**

10834 None.

10835 **4.51.5.3 Environment Variables**

10836 The following environment variables shall affect the execution of pwd:

10837	LANG	This variable shall determine the locale to use for the
10838		locale categories when both LC_ALL and the correspond-
10839		ing environment variable (beginning with LC_) do not
10840		specify a locale. See 2.6.

10841	LC_ALL	This variable shall determine the locale to be used to over-
10842		ride any values for locale categories specified by the set-
10843		tings of LANG or any environment variables beginning
10844		with LC_ .

10845 **LC_MESSAGES** This variable shall determine the language in which mes-
10846 sages should be written.

10847 **4.51.5.4 Asynchronous Events**

10848 Default.

10849 **4.51.6 External Effects**

10850 **4.51.6.1 Standard Output**

10851 The `pwd` utility output shall be an absolute pathname of the current working
10852 directory:

10853 " %s\n", <directory pathname>

10854 **4.51.6.2 Standard Error**

10855 Used only for diagnostic messages.

10856 **4.51.6.3 Output Files**

10857 None.

10858 **4.51.7 Extended Description**

10859 None.

10860 **4.51.8 Exit Status**

10861 The `pwd` utility shall exit with one of the following values:

10862 0 Successful completion.

10863 >0 An error occurred.

10864 **4.51.9 Consequences of Errors**

10865 If an error is detected, output shall not be written to standard output, a diagnos-
10866 tic message shall be written to standard error, and the exit status shall not be
10867 zero.

10868 **4.51.10 Rationale.** *(This subclause is not a part of P1003.2)*

10869 **Examples, Usage**

10870 Some implementations have historically provided `pwd` as a shell special built-in
10871 command.

10872 **History of Decisions Made**

10873 In most utilities, if an error occurs, partial output may be written to standard out-
10874 put. This does not happen in historical implementations of `pwd`. Because `pwd` is
10875 frequently used in existing shell scripts without checking the exit status, it is
10876 important that the historical behavior is required here; therefore, the Conse-
10877 quences of Errors subclause specifically disallows any partial output being writ-
10878 ten to standard output.

10879 **4.52 read — Read a line from standard input**

10880 **4.52.1 Synopsis**

10881 `read [-r] var ...`

10882 **4.52.2 Description**

10883 The `read` utility shall read a single line from standard input.

10884 By default, unless the `-r` option is specified, backslash (`\`) shall act as an escape
10885 character, as described in 3.2.1.

10886 The line shall be split into fields (see the definition in 3.1.3) as in the shell (see
10887 3.6.5); the first field shall be assigned to the first variable *var*, the second field to
10888 the second variable *var*, etc. If there are fewer *var* operands specified than there
10889 are fields, the leftover fields and their intervening separators shall be assigned to
10890 the last *var*. If there are fewer fields than *vars*, the remaining *vars* shall be set to
10891 empty strings.

10892 The setting of variables specified by the *var* operands shall affect the current shell
10893 execution environment; see 3.12.

10894 **4.52.3 Options**

10895 The `read` utility shall conform to the utility argument syntax guidelines
10896 described in 2.10.2.

10897 The following option shall be supported by the implementation:

10898 `-r` Do not treat a backslash character in any special way. Consider
 10899 each backslash to be part of the input line.

10900 **4.52.4 Operands**

10901 The following operands shall be supported by the implementation:

10902 `var` The name of an existing or nonexisting shell variable.

10903 **4.52.5 External Influences**

10904 **4.52.5.1 Standard Input**

10905 The standard input shall be a text file.

10906 **4.52.5.2 Input Files**

10907 None.

10908 **4.52.5.3 Environment Variables**

10909 The following environment variables shall affect the execution of `read`:

10910 **IFS** This variable shall determine the internal field separators
 10911 used to delimit fields. See 3.5.3.

10912 **LANG** This variable shall determine the locale to use for the
 10913 locale categories when both **LC_ALL** and the correspond-
 10914 ing environment variable (beginning with **LC_**) do not
 10915 specify a locale. See 2.6.

10916 **LC_ALL** This variable shall determine the locale to be used to over-
 10917 ride any values for locale categories specified by the set-
 10918 tings of **LANG** or any environment variables beginning
 10919 with **LC_**. 2.6.

10920 **LC_CTYPE** This variable shall determine the locale for the interpreta-
 10921 tion of sequences of bytes of text data as characters (e.g.,
 10922 single- versus multibyte characters in arguments).

10923 **LC_MESSAGES** This variable shall determine the language in which mes-
 10924 sages should be written.

10925 **4.52.5.4 Asynchronous Events**

10926 Default.

10927 **4.52.6 External Effects**

10928 **4.52.6.1 Standard Output**

10929 None.

10930 **4.52.6.2 Standard Error**

10931 Used only for diagnostic messages.

10932 **4.52.6.3 Output Files**

10933 None.

10934 **4.52.7 Extended Description**

10935 None.

10936 **4.52.8 Exit Status**

10937 The `read` utility shall exit with one of the following values:

10938 0 Successful completion.

10939 >0 End-of-file was detected or an error occurred.

10940 **4.52.9 Consequences of Errors**

10941 Default.

10942 **4.52.10 Rationale.** *(This subclause is not a part of P1003.2)*

10943 **Examples, Usage**

10944 The following command:

```
10945     while read -r xx yy
10946     do
10947         printf "%s %s\n" "$yy" "$xx"
10948     done < input_file
```

1

10949 prints a file with the first field of each line moved to the end of the line.

10950 The text in 2.11.5.2 indicates that the results are undefined if an end-of-file is
10951 detected following a backslash at the end of a line when `-r` is not specified.

10952 Since `read` affects the current shell execution environment, it is generally pro-
10953 vided as a shell regular built-in. If it is called in a subshell or separate utility
10954 execution environment, such as one of the following:

1

1

```

10955         (read foo)
10956         nohup read ...
10957         find . -exec read ... \;
10958 it will not affect the shell variables in the caller's environment.

```

10959 **History of Decisions Made**

10960 The `read` utility has historically been a shell built-in. It was separated off into its
 10961 own clause to take advantage of the standard's richer description of functionality
 10962 at the utility level.

10963 The `-r` option was added to enable `read` to subsume the purpose of the historical
 10964 line utility.

10965 **4.53 `rm` — Remove directory entries**

10966 **4.53.1 Synopsis**

```
10967 rm [-fiRr] file ...
```

10968 **4.53.2 Description**

10969 The `rm` utility shall remove the directory entry specified by each *file* argument.

10970 If either of the files `dot` or `dot-dot` are specified as the basename portion of an
 10971 operand (i.e., the final pathname component), `rm` shall write a diagnostic message
 10972 to standard error and do nothing more with such operands.

10973 For each *file* the following steps shall be taken:

- 10974 (1) If the *file* does not exist:
 - 10975 (a) If the `-f` option is not specified, write a diagnostic message to stan-
 10976 dard error.
 - 10977 (b) Go on to any remaining *files*.
- 10978 (2) If *file* is of type directory, the following steps shall be taken:
 - 10979 (a) If neither the `-R` option nor the `-r` option is specified, write a diag-
 10980 nostic message to standard error, do nothing more with *file*, and go
 10981 on to any remaining files.
 - 10982 (b) If the `-f` option is not specified, and either the permissions of *file* do
 10983 not permit writing and the standard input is a terminal or the `-i`
 10984 option is specified, write a prompt to standard error and read a line
 10985 from the standard input. If the response is not affirmative, do noth-
 10986 ing more with the current file and go on to any remaining files.

- 10987 (c) For each entry contained in *file*, other than dot or dot-dot, the four
 10988 steps listed here [(1)-(4)] shall be taken with the entry as if it were a
 10989 *file* operand.
- 10990 (d) If the `-i` option is specified, write a prompt to standard error and
 10991 read a line from the standard input. If the response is not
 10992 affirmative, do nothing more with the current file, and go on to any
 10993 remaining files.
- 10994 (3) If *file* is not of type directory, the `-f` option is not specified, and either the
 10995 permissions of *file* do not permit writing and the standard input is a ter-
 10996 minal or the `-i` option is specified, write a prompt to the standard error
 10997 and read a line from the standard input. If the response is not
 10998 affirmative, do nothing more with the current file and go on to any
 10999 remaining files.
- 11000 (4) If the current file is a directory, `rm` shall perform actions equivalent to
 11001 the POSIX.1 {8} `rmdir()` function called with a pathname of the current
 11002 file used as the *path* argument. If the current file is not a directory, `rm`
 11003 shall perform actions equivalent to the POSIX.1 {8} `unlink()` function
 11004 called with a pathname of the current file used as the *path* argument.
- 11005 If this fails for any reason, `rm` shall write a diagnostic message to stan-
 11006 dard error, do nothing more with the current file, and go on to any
 11007 remaining files.
- 11008 The `rm` utility shall be able to descend to arbitrary depths in a file hierarchy, and
 11009 shall not fail due to path length limitations (unless an operand specified by the
 11010 user exceeds system limitations).

11011 4.53.3 Options

- 11012 The `rm` utility shall conform to the utility argument syntax guidelines described 2
 11013 in 2.10.2. 2
- 11014 The following options shall be supported by the implementation:
- 11015 `-f` Do not prompt for confirmation. Do not write diagnostic mes-
 11016 sages or modify the exit status in the case of nonexistent
 11017 operands. Any previous occurrences of the `-i` option shall be
 11018 ignored.
- 11019 `-i` Prompt for confirmation as described in 4.53.2. Any previous
 11020 occurrences of the `-f` option shall be ignored.
- 11021 `-R` Remove file hierarchies. See 4.53.2.
- 11022 `-r` Equivalent to `-R`.

11023 4.53.4 Operands

11024 The following operand shall be supported by the implementation:

11025 *file* A pathname of a directory entry to be removed.

11026 4.53.5 External Influences

11027 4.53.5.1 Standard Input

11028 Used to read an input line in response to each prompt specified in 4.53.6.1. Oth-
11029 erwise, the standard input shall not be used.

11030 4.53.5.2 Input Files

11031 None.

11032 4.53.5.3 Environment Variables

11033 The following environment variables shall affect the execution of `rm`:

11034	LANG	This variable shall determine the locale to use for the
11035		locale categories when both LC_ALL and the correspond-
11036		ing environment variable (beginning with LC_) do not
11037		specify a locale. See 2.6.
11038	LC_ALL	This variable shall determine the locale to be used to over-
11039		ride any values for locale categories specified by the set-
11040		tings of LANG or any environment variables beginning
11041		with LC_ .
11042	LC_COLLATE	This variable shall determine the locale for the behavior of
11043		ranges, equivalence classes, and multicharacter collating
11044		elements used in the extended regular expression defined
11045		for the <code>yesexpr</code> locale keyword in the LC_MESSAGES
11046		category.
11047	LC_CTYPE	This variable shall determine the locale for the interpreta-
11048		tion of sequences of bytes of text data as characters (e.g.,
11049		single- versus multibyte characters in arguments) and the
11050		behavior of character classes within regular expressions
11051		used in the extended regular expression defined for the
11052		<code>yesexpr</code> locale keyword in the LC_MESSAGES category.
11053	LC_MESSAGES	This variable shall determine the processing of affirmative
11054		responses and the language in which messages should be
11055		written.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

11056 **4.53.5.4 Asynchronous Events**

11057 Default.

11058 **4.53.6 External Effects**

11059 **4.53.6.1 Standard Output**

11060 None.

11061 **4.53.6.2 Standard Error**

11062 Prompts shall be written to standard error under the conditions specified in
11063 4.53.2 and 4.53.3. The prompts shall contain the *file* pathname, but their format
11064 is otherwise unspecified. The standard error shall also be used for diagnostic
11065 messages.

11066 **4.53.6.3 Output Files**

11067 None.

11068 **4.53.7 Extended Description**

11069 None.

11070 **4.53.8 Exit Status**

11071 The `rm` utility shall exit with one of the following values:

11072 0 If the `-f` option was not specified, all the named directory entries were
11073 removed; otherwise, all the existing named directory entries were
11074 removed.

11075 >0 An error occurred.

11076 **4.53.9 Consequences of Errors**

11077 Default.

11078 **4.53.10 Rationale.** *(This subclause is not a part of P1003.2)*

11079 Examples, Usage

11080 The *SVID* requires that systems do not permit the removal of the last link to an
11081 executable binary file that is being executed. Thus, the `rm` utility can fail to
11082 remove such files.

11083 The `-i` option causes `rm` to prompt and read the standard input even if the stan-
11084 dard input is not a terminal, but in the absence of `-i` the mode prompting is not
11085 done when the standard input is not a terminal. 1

11086 For absolute clarity, paragraphs (2)(b) and (3) in 4.53.2, describing `rm`'s behavior
11087 when prompting for confirmation, should be interpreted in the following manner:

```
11088         if ((NOT f_option) AND
11089             ((not_writable AND input_is_terminal) OR i_option))
```

11090 It is forbidden to remove the names `dot` and `dot-dot` in order to avoid the conse-
11091 quences of inadvertently doing something like:

```
11092         rm -r .*
```

11093 The following command

```
11094         rm a.out core
```

11095 removes the directory entries `a.out` and `core`.

11096 The following command

```
11097         rm -Rf junk
```

11098 removes the directory `junk` and all its contents, without prompting.

11099 History of Decisions Made

11100 The exact format of the interactive prompts is unspecified. Only the general
11101 nature of the contents of prompts are specified, because implementations may
11102 desire more descriptive prompts than those used on historical implementations.
11103 Therefore, an application not using the `-f` option, or using the `-i` option relies on
11104 the system to provide the most suitable dialogue directly with the user, based on
11105 the behavior specified.

11106 The `-r` option is existing practice on all known systems. The synonym `-R` option
11107 is provided for consistency with the other utilities in this standard that provide
11108 options requesting recursive descent.

11109 The behavior of the `-f` option in historical versions of `rm` is inconsistent. In gen-
11110 eral, along with “forcing” the unlink without prompting for permission, it always
11111 causes diagnostic messages to be suppressed and the exit status to be unmodified
11112 for nonexistent operands and files that cannot be unlinked. In some versions,
11113 however, the `-f` option suppresses usage messages and system errors as well.
11114 Suppressing such messages is not a service to either shell scripts or users.

11115 It is less clear that error messages regarding unlinkable files should be
 11116 suppressed. Although this is historical practice, this standard does not permit
 11117 the `-f` option to suppress such messages.

11118 When given the `-r` and `-i` options, historical versions of `rm` prompt the user twice
 11119 for each directory, once before removing its contents and once before actually
 11120 attempting to delete the directory entry that names it. This allows the user to
 11121 “prune” the file hierarchy walk. Historical versions of `rm` were inconsistent in
 11122 that some did not do the former prompt for directories named on the command
 11123 line and others had obscure prompting behavior when the `-i` option was specified
 11124 and the permissions of the file did not permit writing. The POSIX.2 `rm` differs lit-
 11125 tle from historic practice, but does require that prompts be consistent. Historical
 11126 versions of `rm` were also inconsistent in that prompts were done to both standard
 11127 output and standard error. POSIX.2 requires that prompts be done to standard
 11128 error, for consistency with `cp` and `mv` and to allow existing extensions to `rm` that
 11129 provide an option to list deleted files on standard output.

11130 The `rm` utility is required to descend to arbitrary depths so that any file hierarchy
 11131 may be deleted. This means, for example, that the `rm` utility cannot run out of file
 11132 descriptors during its descent, i.e., if the number of file descriptors is limited, `rm`
 11133 cannot be implemented in the historical fashion where a file descriptor is used per
 11134 directory level. Also, `rm` is not permitted to fail because of path length restric-
 11135 tions, unless an operand specified by the user is longer than `{PATH_MAX}`.

11136 4.54 `rmdir` — Remove directories

11137 4.54.1 Synopsis

11138 `rmdir [-p] dir...`

11139 4.54.2 Description

11140 The `rmdir` utility shall remove the directory entry specified by each *dir* operand,
 11141 which shall refer to an empty directory.

11142 Directories shall be processed in the order specified. If a directory and a subdirec-
 11143 tory of that directory are specified in a single invocation of the `rmdir` utility, the
 11144 subdirectory shall be specified before the parent directory so that the parent direc-
 11145 tory will be empty when the `rmdir` utility tries to remove it.

11146 4.54.3 Options

11147 The `rmdir` utility shall conform to the utility argument syntax guidelines
11148 described in 2.10.2.

11149 The following option shall be supported by the implementation:

- 11150 `-p` Remove all directories in a pathname. For each *dir* operand:
- 11151 (1) The directory entry it names shall be removed.
- 11152 (2) If the *dir* operand includes more than one pathname com-
11153 ponent, effects equivalent to the following command shall
11154 occur:
- 11155 `rmdir -p $(dirname dir)`

11156 4.54.4 Operands

11157 The following operand shall be supported by the implementation:

11158 *dir* A pathname of an empty directory to be removed.

11159 4.54.5 External Influences

11160 4.54.5.1 Standard Input

11161 None.

11162 4.54.5.2 Input Files

11163 None.

11164 4.54.5.3 Environment Variables

11165 The following environment variables shall affect the execution of `rmdir`:

11166	LANG	This variable shall determine the locale to use for the
11167		locale categories when both LC_ALL and the correspond-
11168		ing environment variable (beginning with LC_) do not
11169		specify a locale. See 2.6.
11170	LC_ALL	This variable shall determine the locale to be used to over-
11171		ride any values for locale categories specified by the set-
11172		tings of LANG or any environment variables beginning
11173		with LC_ .
11174	LC_CTYPE	This variable shall determine the locale for the interpreta-
11175		tion of sequences of bytes of text data as characters (e.g.,
11176		single- versus multibyte characters in arguments).

11177 **LC_MESSAGES** This variable shall determine the language in which mes-
 11178 sages should be written.

11179 **4.54.5.4 Asynchronous Events**

11180 Default.

11181 **4.54.6 External Effects**

11182 **4.54.6.1 Standard Output**

11183 None.

11184 **4.54.6.2 Standard Error**

11185 Used only for diagnostic messages.

11186 **4.54.6.3 Output Files**

11187 None.

11188 **4.54.7 Extended Description**

11189 None.

11190 **4.54.8 Exit Status**

11191 The `rmdir` utility shall exit with one of the following values:

11192	0	Each directory entry specified by a <i>dir</i> operand was removed success-
11193		fully.
11194	>0	An error occurred.

11195 **4.54.9 Consequences of Errors**

11196 Default.

11197 **4.54.10 Rationale.** (*This subclause is not a part of P1003.2*)

11198 **Examples, Usage**

11199 On historical System V systems, the `-p` option also caused a message to be writ-
 11200 ten to the standard output. The message indicated whether the whole path was
 11201 removed or part of the path remains for some reason. The Standard Error sub-
 11202 clause requires this diagnostic when the entire path specified by a *dir* operand is
 11203 not removed, but does not allow the status message reporting success to be writ-
 11204 ten as a diagnostic.

11205 If a directory *a* in the current directory is empty except it contains a directory *b*
 11206 and *a/b* is empty except it contains a directory *c*,

11207 `rmkdir -p a/b/c`

11208 will remove all three directories.

11209 The `rmkdir` utility on System V also included an `-s` option that suppressed the
 11210 informational message output by the `-p` option. This option has been omitted
 11211 because the informational message is not specified by POSIX.2.

11212 **4.55 sed — Stream editor**

11213 **4.55.1 Synopsis**

11214 `sed [-n] script [file ...]`

11215 `sed [-n] [-e script] ... [-f script_file] ... [file ...]`

11216 **4.55.2 Description**

11217 The `sed` utility is a stream editor that shall read one or more text files, make edit-
 11218 ing changes according to a script of editing commands, and write the results to
 11219 standard output. The script shall be obtained from either the *script* operand
 11220 string or a combination of the option-arguments from the `-e script` and
 11221 `-f script_file` options.

11222 **4.55.3 Options**

11223 The `sed` utility shall conform to the utility argument syntax guidelines described
 11224 in 2.10.2, except that the order of presentation of the `-e` and `-f` options is
 11225 significant.

11226 The following options shall be supported by the implementation:

- 11227 `-e script` Add the editing commands specified by the *script* option-
 11228 argument to the end of the script of editing commands. The *script*
 11229 option-argument shall have the same properties as the *script*
 11230 operand, described in 4.55.4.
- 11231 `-f script_file` Add the editing commands in the file *script_file* to the end of the
 11232 script.
 11233
- 11234 `-n` Suppress the default output (in which each line, after it is exam-
 11235 ined for editing, is written to standard output). Only lines expli-
 11236 citedly selected for output shall be written.
- 11237 Multiple `-e` and `-f` options may be specified. All commands shall be added to the
 11238 script in the order specified, regardless of their origin.

11239 **4.55.4 Operands**

11240 The following operands shall be supported by the implementation:

- 11241 *file* A pathname of a file whose contents shall be read and edited. If
 11242 multiple *file* operands are specified, the named files shall be read
 11243 in the order specified and the concatenation shall be edited. If no
 11244 *file* operands are specified, the standard input shall be used.
- 11245 *script* A string to be used as the script of editing commands. The appli-
 11246 cation shall not present a *script* that violates the restrictions of a
 11247 text file (see 2.2.2.151), except that the final character need not be
 11248 a <newline>.

11249 **4.55.5 External Influences**

11250 **4.55.5.1 Standard Input**

- 11251 The standard input shall be used only if no *file* operands are specified. See Input
 11252 Files.

11253 **4.55.5.2 Input Files**

- 11254 The input files shall be text files. The *script_files* named by the `-f` option shall
 11255 consist of editing commands, one per line.

11256 **4.55.5.3 Environment Variables**

- 11257 The following environment variables shall affect the execution of `sed`:

11258	LANG	This variable shall determine the locale to use for the
11259		locale categories when both LC_ALL and the correspond-
11260		ing environment variable (beginning with LC_) do not
11261		specify a locale. See 2.6.
11262	LC_ALL	This variable shall determine the locale to be used to over-
11263		ride any values for locale categories specified by the set-
11264		tings of LANG or any environment variables beginning
11265		with LC_ .
11266	LC_COLLATE	This variable shall determine the locale for the behavior of
11267		ranges, equivalence classes, and multicharacter collating
11268		elements within regular expressions.
11269	LC_CTYPE	This variable shall determine the locale for the interpreta-
11270		tion of sequences of bytes of text data as characters (e.g.,
11271		single- versus multibyte characters in arguments and
11272		input files), and the behavior of character classes within
11273		regular expressions.
11274	LC_MESSAGES	This variable shall determine the language in which mes-
11275		sages should be written.

11276 **4.55.5.4 Asynchronous Events**

11277 Default.

11278 **4.55.6 External Effects**

11279 **4.55.6.1 Standard Output**

11280 The input files shall be written to standard output, with the editing commands
 11281 specified in the script applied. If the **-n** option is specified, only those input lines
 11282 selected by the script shall be written to standard output.

11283 **4.55.6.2 Standard Error**

11284 Used only for diagnostic messages.

11285 **4.55.6.3 Output Files**

11286 The output files shall be text files whose formats are dependent on the editing
 11287 commands given.

4.55.7 Extended Description

The *script* shall consist of editing commands, one per line, of the following form:

[*address*[,*address*]]*command*[*arguments*]

Zero or more <blank>s shall be accepted before the first address and before *command*.

In default operation, *sed* cyclically shall copy a line of input, less its terminating <newline>, into a *pattern space* (unless there is something left after a D command), apply in sequence all commands whose addresses select that pattern space, and at the end of the script copy the pattern space to standard output (except when *-n* is specified) and delete the pattern space. Whenever the pattern space is written to standard output or a named file, *sed* shall immediately follow it with a <newline>.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval. The *pattern* and *hold spaces* shall each be able to hold at least 8192 bytes.

4.55.7.1 *sed* Addresses

An address is either empty, a decimal number that counts input lines cumulatively across files, a \$ character that addresses the last line of input, or a context address (which consists of a regular expression as described in 4.55.7.2, preceded and followed by a delimiter, usually a slash).

A command line with no addresses shall select every pattern space.

A command line with one address shall select each pattern space that matches the address.

A command line with two addresses shall select the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line shall be selected.) Starting at the first line following the selected range, *sed* shall look again for the first address. Thereafter the process shall be repeated.

Editing commands can be applied only to nonselected pattern spaces by use of the negation command ! (see 4.55.7.3).

4.55.7.2 *sed* Regular Expressions

The *sed* utility shall support the basic regular expressions described in 2.8.3, with the following additions:

- (1) In a context address, the construction *\cREc*, where *c* is any character other than <backslash> or <newline>, shall be identical to */RE/*. If the character designated by *c* appears following a backslash, then it shall be considered to be that literal character, which shall not terminate the RE. For example, in the context address *\xabc\xdefx*, the second x

11327 stands for itself, so that the regular expression is `abcxdef`.
 11328 (2) The escape sequence `\n` shall match a `<newline>` embedded in the pat-
 11329 tern space. A literal `<newline>` character shall not be used in the regu-
 11330 lar expression of a context address or in the substitute command.

11331 4.55.7.3 `sed` Editing Commands

11332 In the following list of commands, the maximum number of permissible addresses
 11333 for each command is indicated by `[0addr]`, `[1addr]`, or `[2addr]`, representing
 11334 zero, one, or two addresses.

11335 The argument *text* shall consist of one or more lines. Each embedded `<newline>`
 11336 in the text shall be preceded by a backslash. Other backslashes in text shall be
 11337 removed and the following character shall be treated literally.

11338 The `r` and `w` commands take an optional *rfile* (or *wfile*) parameter, separated from
 11339 the command letter by one or more `<blank>`s; implementations may allow zero
 11340 separation as an extension.

11341 The argument *rfile* or the argument *wfile* shall terminate the command line. Each
 11342 *wfile* shall be created before processing begins. Implementations shall support at
 11343 least nine *wfile* arguments in the script; the actual number (≥ 9) that shall be sup-
 11344 ported by the implementation is unspecified. The use of the *wfile* parameter shall
 11345 cause that file to be initially created, if it does not exist, or shall replace the con-
 11346 tents of an existing file.

11347 The `b`, `r`, `s`, `t`, `w`, `y`, `!`, and `:` commands shall accept additional arguments. The
 11348 following synopses indicate which arguments shall be separated from the com-
 11349 mands by a single `<space>`.

11350 Two of the commands take a *command-list*, which is a list of `sed` commands
 11351 separated by `<newline>`s, as follows:

```
11352     { command
11353     command
11354     . . .
11355     }
```

11356 The `{` can be preceded with `<blank>`s and can be followed with white space. The
 11357 *commands* can be preceded by white space. The terminating `}` shall be preceded
 11358 by a `<newline>` and then zero or more `<blank>`s.

11359 `[2addr] {command-list`
 11360 `}` Execute *command-list* only when the pattern space is selected.

11361 `[1addr]a\`
 11362 *text* Write *text* to standard output just before each attempt to fetch a 1
 11363 line of input, whether by executing the `N` command or by begin- 1
 11364 ning a new cycle. 1

11365 `[2addr]b [label]`
 11366 Branch to the `:` command bearing the *label*. If *label* is not
 11367 specified, branch to the end of the script. The implementation

11368		shall support <i>labels</i> recognized as unique up to at least 8 characters; the actual length (≥ 8) that shall be supported by the implementation is unspecified. It is unspecified whether exceeding a label length causes an error or a silent truncation.	
11369			
11370			
11371			
11372	[2addr]c\		
11373	text	Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place <i>text</i> on the output.	
11374			
11375	[2addr]d	Delete the pattern space and start the next cycle.	
11376	[2addr]D	Delete the initial segment of the pattern space through the first <newline> and start the next cycle.	
11377			
11378	[2addr]g	Replace the contents of the pattern space by the contents of the hold space.	
11379			
11380	[2addr]G	Append to the pattern space a <newline> followed by the contents of the hold space.	1
11381			1
11382	[2addr]h	Replace the contents of the hold space with the contents of the pattern space.	
11383			
11384	[2addr]H	Append to the hold space a <newline> followed by the contents of the pattern space.	1
11385			1
11386	[1addr]i\		
11387	text	Write <i>text</i> to standard output.	1
11388	[2addr]l	(The letter ell.) Write the pattern space to standard output in a visually unambiguous form. The characters listed in Table 2-15 (see 2.12) shall be written as the corresponding escape sequence. Nonprintable characters not in Table 2-15 shall be written as one three-digit octal number (with a preceding <backslash>) for each byte in the character (most significant byte first). If the size of a byte on the system is greater than nine bits, the format used for nonprintable characters is implementation defined.	1
11389			1
11390			1
11391			1
11392			1
11393			1
11394			1
11395			1
11396		Long lines shall be folded, with the point of folding indicated by writing <backslash><newline>; the length at which folding occurs is unspecified, but should be appropriate for the output device. The end of each line shall be marked with a \$.	1
11397			1
11398			1
11399			1
11400	[2addr]n	Write the pattern space to standard output if the default output has not been suppressed, and replace the pattern space with the next line of input.	
11401			
11402			
11403	[2addr]N	Append the next line of input to the pattern space, using an embedded <newline> to separate the appended material from the original material. Note that the current line number changes.	
11404			
11405			

11406	[2addr]p	Write the pattern space to standard output.	
11407	[2addr]P	Write the pattern space, up to the first <newline>, to standard	1
11408		output.	
11409	[1addr]q	Branch to the end of the script and quit without starting a new	
11410		cycle.	
11411	[1addr]r <i>rfile</i>		
11412		Copy the contents of <i>rfile</i> to standard output just before each	1
11413		attempt to fetch a line of input. If <i>rfile</i> does not exist or cannot be	1
11414		read, it shall be treated as if it were an empty file, causing no	1
11415		error condition.	1
11416	[2addr]s / <i>regular expression</i> / <i>replacement</i> / <i>flags</i>		
11417		Substitute the <i>replacement</i> string for instances of the <i>regular</i>	
11418		<i>expression</i> in the pattern space. Any character other than	1
11419		<backslash> or <newline> can be used instead of a slash to	1
11420		delimit the RE and the replacement. Within the RE and the	1
11421		replacement, the RE delimiter itself can be used as a literal char-	
11422		acter if it is preceded by a backslash.	
11423		An ampersand (&) appearing in the <i>replacement</i> shall be replaced	
11424		by the string matching the RE. The special meaning of & in this	
11425		context can be suppressed by preceding it by backslash. The	
11426		characters \n, where <i>n</i> is a digit, shall be replaced by the text	
11427		matched by the corresponding backreference expression (see	
11428		2.8.3.3).	
11429		A line can be split by substituting a <newline> character into it.	1
11430		The application shall escape the <newline> in the <i>replacement</i>	1
11431		by preceding it by backslash. A substitution shall be considered	1
11432		to have been performed even if the replacement string is identical	
11433		to the string that it replaces.	
11434		The value of <i>flags</i> shall be zero or more of:	
11435	<i>n</i>	Substitute for the <i>n</i> th occurrence only of the <i>regular</i>	
11436		<i>expression</i> found within the pattern space.	
11437	<i>g</i>	Globally substitute for all nonoverlapping instances	
11438		of the <i>regular expression</i> rather than just the first	
11439		one. If both <i>g</i> and <i>n</i> are specified, the results are	
11440		unspecified.	
11441	<i>p</i>	Write the pattern space to standard output if a	
11442		replacement was made.	
11443	<i>w wfile</i>	Write. Append the pattern space to <i>wfile</i> if a	
11444		replacement was made.	
11445	[2addr]t [<i>label</i>]		
11446		Test. Branch to the : command bearing the <i>label</i> if any substitu-	
11447		tions have been made since the most recent reading of an input	

11448		line or execution of a <code>t</code> . If <i>label</i> is not specified, branch to the end	
11449		of the script.	
11450	<code>[2addr]w</code>	<i>wfile</i>	
11451		Append [write] the pattern space to <i>wfile</i> .	
11452	<code>[2addr]x</code>	Exchange the contents of the pattern and hold spaces.	
11453	<code>[2addr]y/string1/string2/</code>		
11454		Replace all occurrences of characters in <i>string1</i> with the	
11455		corresponding characters in <i>string2</i> . If the number of characters	
11456		in <i>string1</i> and <i>string2</i> are not equal, or if any of the characters in	
11457		<i>string1</i> appear more than once, the results are undefined. Any	
11458		character other than <code><backslash></code> or <code><newline></code> can be used	1
11459		instead of slash to delimit the strings. Within <i>string1</i> and	1
11460		<i>string2</i> , the delimiter itself can be used as a literal character if it	1
11461		is preceded by a backslash.	1
11462	<code>[2addr]!command</code>		
11463	<code>[2addr]!{command-list</code>		
11464	<code>}</code>	Apply the <i>command</i> or <i>command-list</i> only to the lines that are not	
11465		selected by the address(es).	
11466	<code>[0addr]:label</code>		
11467		This command shall do nothing; it bears a <i>label</i> for the <code>b</code> and <code>t</code>	
11468		commands to branch to.	
11469	<code>[1addr]=</code>	Write the following to standard output:	
11470		<code>"%d\n", <current line number></code>	1
11471	<code>[0addr]</code>	An empty command shall be ignored.	
11472	<code>[0addr]#</code>	The <code>#</code> and the remainder of the line shall be ignored (treated as a	
11473		comment), with the single exception that if the first two charac-	
11474		ters in the file are <code>#n</code> , the default output shall be suppressed; this	
11475		shall be the equivalent of specifying <code>-n</code> on the command line.	

11476 4.55.8 Exit Status

11477 The `sed` utility shall exit with one of the following values:

11478	0	Successful completion.
11479	>0	An error occurred.

11480 4.55.9 Consequences of Errors

11481 Default.

11482 4.55.10 Rationale. *(This subclause is not a part of P1003.2)*

11483 Examples, Usage

11484 See the rationale for `cat` (4.4.10) for an example `sed` script.

11485 This standard requires implementations to support at least nine distinct *wfiles*,
11486 matching historical practice on many implementations. Implementations are
11487 encouraged to support more, but portable applications should not exceed this
11488 limit.

11489 Note that regular expressions match entire strings, not just individual lines, but
11490 `<newline>` is matched by `\n` in a `sed` RE; `<newline>` is not allowed in an RE.
11491 Also note that `\n` cannot be used to match a `<newline>` at the end of an input
11492 line; `<newline>`s appear in the pattern space as a result of the `N` editing com-
11493 mand.

11494 The exit status codes specified here are different from those in System V.
11495 System V returns 2 for garbled `sed` commands, but returns zero with its usage
11496 message or if the input file could not be opened. The working group considered
11497 this to be a bug.

11498 History of Decisions Made

11499 The manner in which the `l` command writes nonprintable characters was
11500 changed to avoid the historical backspace-overstrike method and added other 1
11501 requirements to achieve unambiguous output. See the rationale for `ed` (4.20.10) 1
11502 for details of the format chosen, which is the same as that chosen for `sed`. 1

11503 The standard requires implementations to provide pattern and hold spaces of at
11504 least 8192 bytes, larger than the 4000-byte spaces used by some historical imple-
11505 mentations, but less than the 20K byte limit used in an earlier draft. Implemen-
11506 tations are encouraged to dynamically allocate larger pattern and hold spaces as
11507 needed.

11508 The requirements for acceptance of `<blank>`s and `<space>`s in command lines
11509 has been made more explicit than in earlier drafts to clearly describe existing
11510 practice and remove confusion about the phrase “protect initial blanks [sic] and
11511 tabs from the stripping that is done on every script line” that appears in much of
11512 the historical documentation of the `sed` utility description of text. (Not all imple- 1
11513 mentations are known to have stripped `<blank>`s from text lines, although they 1
11514 all have allowed leading `<blank>`s preceding the address on a command line.) 1

11515 The treatment of `#` comments differs from the *SVID*, which only allows a comment
11516 as the first line of the script, but matches BSD-derived implementations. The
11517 comment character is treated as a command and it has the same properties in
11518 terms of being accepted with leading `<blank>`s; the BSD implementation has

11519 historically supported this.

11520 Earlier drafts of POSIX.2 required that a *script_file* have at least one noncomment
 11521 line. Some historical implementations have behaved in unexpected ways if this
 11522 were not the case. The working group felt that this was incorrect behavior, and
 11523 that application developers should not have to work around this feature. A
 11524 correct implementation of POSIX.2 shall permit *script_files* that consist only of
 11525 comment lines.

11526 Earlier drafts indicated that if `-e` and `-f` options were intermixed, all `-e` options
 11527 were processed before any `-f` options. This has been changed to process them in
 11528 the order presented because it matches existing practice and is more intuitive.

11529 The treatment of the `p` flag to the `s` command differs between System V and BSD-
 11530 based systems (actually, between Version 7 and 32V) when the default output is
 11531 suppressed. In the two examples:

```
11532      echo a | sed      's/a/A/p'
11533      echo a | sed -n  's/a/A/p'
```

11534 POSIX.2, BSD, System V documentation, and the *SVID* indicate that the first
 11535 example should write two lines with `A`, whereas the second should write one.
 11536 Some System V systems write the `A` only once in both examples, because the `p`
 11537 flag is ignored if the `-n` option is not specified.

11538 This is a case of a diametrical difference between systems that could not be recon-
 11539 ciled through the compromise of declaring the behavior to be unspecified. The
 11540 *SVID*/BSD/32V behavior was adopted for POSIX.2 because:

- 11541 — No known documentation for any historic system describes the interaction
 11542 between the `p` flag and the `-n` option.
- 11543 — The selected behavior is more correct as there is no technical justification
 11544 for any interaction between the `p` flag and the `-n` option. A relationship
 11545 between `-n` and the `p` flag might imply that they are only used together
 11546 (when `p` should be a no-op), but this ignores valid scripts that interrupt the
 11547 cyclical nature of the processing through the use of the `D`, `d`, `q`, or branching
 11548 commands. Such scripts rely on the `p` suffix to write the pattern space
 11549 because they do not make use of the default output at the “bottom” of the
 11550 script.
- 11551 — Because the `-n` option makes the `p` flag a no-op, any interaction would only
 11552 be useful if `sed` scripts were written to run both with and without the `-n`
 11553 option. This is believed to be unlikely. It is even more unlikely that pro-
 11554 grammers have coded the `p` flag expecting it to be a no-op. Because the
 11555 interaction was not documented, the likelihood of a programmer discover-
 11556 ing the interaction and depending on it is further decreased.
- 11557 — Finally, scripts that break under the specified behavior will produce too
 11558 much output instead of too little, which is easier to diagnose and correct.

11559 The form of the substitute command that uses the `n` suffix was limited to the first
 11560 512 matches in a previous draft. This limit has been removed because there is no
 11561 reason an editor processing lines of `{LINE_MAX}` length should have this

11562 restriction. The command `s/a/A/2047` should be able to substitute the 2047th
 11563 occurrence of `a` on a line.

11564 **4.56 sh — Shell, the standard command language interpreter**

11565 **4.56.1 Synopsis**

```
11566 sh [-aCefinuvx] [ command_file [argument ...] ] 1
11567 sh -c [-aCefinuvx] command_string [ command_name [argument ...] ] 1
11568 sh -s [-aCefinuvx] [argument ...] 1
```

11569 **4.56.2 Description**

11570 The `sh` utility is a command language interpreter that shall execute commands
 11571 read from a command-line string, the standard input, or a specified file. The com-
 11572 mands to be executed shall be expressed in the language described in Section 3.

11573 **4.56.3 Options**

11574 The `sh` utility shall conform to the utility argument syntax guidelines described
 11575 in 2.10.2.

11576 The `-a`, `-C`, `-e`, `-f`, `-n`, `-u`, `-v`, and `-x` options are described as part of the `set` util-
 11577 ity in 3.14.11. The following additional options shall be supported by the imple-
 11578 mentation:

11579	<code>-c</code>	Read commands from the <i>command_string</i> operand. Set the
11580		value of special parameter 0 (see 3.5.2) from the value of the
11581		<i>command_name</i> operand and the positional parameters (<code>\$1</code> , <code>\$2</code> ,
11582		etc.) in sequence from the remaining <i>argument</i> operands. No
11583		commands shall be read from the standard input.
11584	<code>-i</code>	Specify that the shell is <i>interactive</i> ; see below. An implementa-
11585		tion may treat specifying the <code>-i</code> option as an error if the real user
11586		ID of the calling process does not equal the effective user ID or if
11587		the real group ID does not equal the effective user ID.
11588	<code>-s</code>	Read commands from the standard input.

11589 If there are no operands and the `-c` option is not specified, the `-s` option shall be
 11590 assumed.

11591 If the `-i` option is present, or if there are no operands and the shell's standard
 11592 input and standard error are attached to a terminal, the shell is considered to be
 11593 *interactive*. (See 3.1.4.) The behavior of an interactive shell is not fully specified
 11594 by this standard.

11595 NOTE: The preceding sentence is expected to change following the eventual approval of the UPE
11596 supplement.

11597 Implementations may accept the option letters with a leading plus sign (+)
11598 instead of a leading hyphen (meaning the reverse case of the option as described
11599 in this standard). A conforming application shall protect its first operand, if it
11600 starts with a plus sign, by preceding it with the -- argument that denotes “end of
11601 options.”

11602 4.56.4 Operands

11603 The following operands shall be supported by the implementation:

- 11604 - A single hyphen shall be treated as the first operand and then
11605 ignored. If both - and -- are given as arguments, or if other
11606 operands precede the single hyphen, the results are undefined.
- 11607 *argument* The positional parameters (\$1, \$2, etc.) shall be set to *argu-*
11608 *ments*, if any.
- 11609 *command_file*
11610 The pathname of a file containing commands. If the pathname 1
11611 contains one or more slash characters, the implementation shall 1
11612 attempt to read that file; the file need not be executable. If the 1
11613 pathname does not contain a slash character:
 - 11614 — The implementation shall attempt to read that file from the
11615 current working directory; the file need not be executable.
 - 11616 — If the file is not in the current working directory, the imple-
11617 mentation may perform a search for an executable file using
11618 the value of **PATH**, as described in 3.9.1.1.
- 11619 Special parameter 0 (see 3.5.2) shall be set to the value of
11620 *command_file*. If *sh* is called using a synopsis form that omits
11621 *command_file*, special parameter 0 shall be set to the value of the
11622 first argument passed to *sh* from its parent (e.g., *argv*[0] in the C
11623 binding), which is normally a pathname used to execute the *sh*
11624 utility.
- 11625 *command_name*
11626 A string assigned to special parameter 0 when executing the com-
11627 mands in *command_string*. If *command_name* is not specified,
11628 special parameter 0 shall be set to the value of the first argument
11629 passed to *sh* from its parent (e.g., *argv*[0] in the C binding), which
11630 is normally a pathname used to execute the *sh* utility.
- 11631 *command_string*
11632 A string that shall be interpreted by the shell as one or more com-
11633 mands, as if the string were the argument to the function in 7.1.1
11634 [such as the *system()* function in the C binding]. If the 1
11635 *command_string* operand is an empty string, *sh* shall exit with a 1

11636 zero exit status. 1

11637 4.56.5 External Influences

11638 4.56.5.1 Standard Input

11639 The standard input shall be used only if:

- 11640 (1) The `-s` option is specified, or;
- 11641 (2) The `-c` option is not specified and no operands are specified, or;
- 11642 (3) The script executes one or more commands that require input from stan-
11643 dard input (such as a `read` command that does not redirect its input).

11644 See Input Files.

11645 When the shell is using standard input and it invokes a command that also uses
11646 standard input, the shell shall ensure that the standard input file pointer points
11647 directly after the command it has read when the command begins execution. It 1
11648 shall not read ahead in such a manner that any characters intended to be read by 1
11649 the invoked command are consumed by the shell (whether interpreted by the shell 1
11650 or not) or that characters that are not read by the invoked command are not seen 1
11651 by the shell. When the command expecting to read standard input is started
11652 asynchronously by an interactive shell, it is unspecified whether characters are
11653 read by the command or interpreted by the shell.

11654 If the standard input to `sh` is a FIFO or terminal device and is set to nonblocking 1
11655 reads, then `sh` shall enable blocking reads on standard input. This shall remain 1
11656 in effect when the command completes. 1

11657 4.56.5.2 Input Files

11658 The input file shall be a text file, except that line lengths shall be unlimited. If 1
11659 the input file is empty or consists solely of blank lines and/or comments, `sh` shall 1
11660 exit with a zero exit status. 1

11661 4.56.5.3 Environment Variables

11662 The following environment variables shall affect the execution of `sh`:

11663	HOME	This variable shall be interpreted as the pathname of the
11664		user's home directory. The contents of HOME are used in
11665		Tilde Expansion as described in 3.6.1.
11666	IFS	<i>Input field separators</i> : a string treated as a list of charac-
11667		ters that shall be used for field splitting and to split lines
11668		into words with the <code>read</code> command. See 3.6.5. If IFS is
11669		not set, the shell shall behave as if the value of IFS were
11670		the <code><space></code> , <code><tab></code> , and <code><newline></code> characters. Imple-
11671		mentations may ignore the value of IFS in the

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

11672		environment at the time <code>sh</code> is invoked, treating IFS as if
11673		it were not set.
11674	LANG	This variable shall determine the locale to use for the
11675		locale categories when both LC_ALL and the correspond-
11676		ing environment variable (beginning with LC_) do not
11677		specify a locale. See 2.6.
11678	LC_ALL	This variable shall determine the locale to be used to over-
11679		ride any values for locale categories specified by the set-
11680		tings of LANG or any environment variables beginning
11681		with LC_ .
11682	LC_COLLATE	This variable shall determine the behavior of range
11683		expressions, equivalence classes, and multicharacter col-
11684		lating elements within pattern matching.
11685	LC_CTYPE	This variable shall determine the locale for the interpreta-
11686		tion of sequences of bytes of text data as characters (e.g.,
11687		single- versus multibyte characters in arguments and
11688		input files), which characters are defined as letters (char-
11689		acter class <code>alpha</code>), and the behavior of character classes
11690		within pattern matching.
11691	LC_MESSAGES	This variable shall determine the language in which mes-
11692		sages should be written.
11693	PATH	This variable shall represent a string formatted as
11694		described in 2.6, used to effect command interpretation.
11695		See 3.9.1.1.

11696 **4.56.5.4 Asynchronous Events**

11697 Default.

11698 **4.56.6 External Effects**

11699 **4.56.6.1 Standard Output**

11700 See Standard Error.

11701 **4.56.6.2 Standard Error**

11702 Except as otherwise stated (by the descriptions of any invoked utilities or in
11703 interactive mode), standard error is used only for diagnostic messages.

11704 **4.56.6.3 Output Files**

11705 None.

11706 **4.56.7 Extended Description**

11707 See Section 3.

11708 **4.56.8 Exit Status**11709 The `sh` utility shall exit with one of the following values: 1

11710 0 The script to be executed consisted solely of zero or more blank lines 1
 11711 and/or comments. 1

11712 1–125 A noninteractive shell detected a syntax, redirection, or variable 1
 11713 assignment error. 1

11714 127 A specified *command_file* could not be found by a noninteractive 1
 11715 shell. 1

11716 Otherwise, the shell shall return the exit status of the last command it invoked or
 11717 attempted to invoke (see also the `exit` utility in 3.14.7).

11718 **4.56.9 Consequences of Errors**

11719 See 3.8.1.

11720 **4.56.10 Rationale.** (*This subclause is not a part of P1003.2*)11721 **Examples, Usage**11722 `sh -c "cat myfile"`11723 `sh my_shell_cmds`

11724 The `sh` utility and the `set` special built-in utility share a common set of options.
 11725 Unlike `set`, however, the POSIX.2 `sh` does not specify the use of `+` as an option
 11726 flag, because it is not particularly useful (the `+` variety generally invokes the
 11727 default behavior) and because `getopt()` does not support it. However, since many
 11728 historical implementations do support the plus, applications will have to guard
 11729 against the relatively obscure case of a first operand with a leading plus sign.

11730 There is a large number of environment variables used by historical implementa-
 11731 tions of `sh` that will not be introduced by POSIX.2 until the UPE is completed.

11732 The KornShell ignores the contents of **IFS** upon entry to the script. A conforming
 11733 application cannot rely on importing **IFS**. One justification for this, beyond secu-
 11734 rity considerations, is to assist possible future shell compilers. Allowing **IFS** to be
 11735 imported from the environment will prevent many optimizations that might oth-
 11736 erwise be performed via dataflow analysis of the script itself.

11737 The standard input and standard error are the files that determine whether a
 11738 shell is interactive when `-i` is not specified. For example,

11739 `sh > file` and `sh 2> file`

11740 create interactive and noninteractive shells, respectively. Although both accept
 11741 terminal input, the results of error conditions will be different, as described in
 11742 3.8.1; in the second example a redirection error encountered by a special built-in
 11743 utility will abort the shell.

11744 The text in Standard Input about nonblocking reads concerns an instance of `sh` 1
 11745 that has been invoked, probably by a C-language program, with standard input 1
 11746 that has been opened using the `O_NONBLOCK` flag; see POSIX.1 {8} `open()`. If the 1
 11747 shell did not reset this flag, it would immediately terminate because no input data 1
 11748 would be available yet and that would be considered the same as end-of-file. 1

11749 **History of Decisions Made**

11750 See the Rationale for Section 3 concerning the lack of interactive features in `sh`.
 11751 These features, including optional job control, are scheduled to be added in the
 11752 User Portability Extension.

11753 The **PS1** and **PS2** variables are not specified because this standard, without UPE,
 11754 does not describe an interactive shell.

11755 The options associated with a *restricted shell* (command name `rsh` and the `-r`
 11756 option) were excluded because the developers of the standard felt that the implied
 11757 level of security was not achievable and they did not want to raise false expecta-
 11758 tions.

11759 On systems that support set-user-ID scripts, a historical trapdoor has been to link
 11760 a script to the name `-i`. When it is called by a sequence such as `sh -` or by
 11761 `#!/bin/sh -` the historical systems have assumed that no option letters follow.
 11762 Thus, POSIX.2 allows the single hyphen to mark the end of the options, in addi-
 11763 tion to the use of the regular `--` argument, because it was felt that the older prac-
 11764 tice was so pervasive. An alternative approach is taken by the KornShell, where
 11765 real and effective user/group IDs must match for an interactive shell; this
 11766 behavior is specifically allowed by POSIX.2. (Note: there are other problems with
 11767 set-user-ID scripts that the two approaches described here do not deal with.)

11768 **4.57 sleep — Suspend execution for an interval**

11769 **4.57.1 Synopsis**

11770 `sleep time`

11771 **4.57.2 Description**

11772 The `sleep` utility shall suspend execution for at least the integral number of
11773 seconds specified by the *time* operand.

11774 **4.57.3 Options**

11775 None.

11776 **4.57.4 Operands**

11777 The following operands shall be supported by the implementation:

11778	<i>time</i>	A nonnegative decimal integer specifying the number of seconds
11779		for which to suspend execution.

11780 **4.57.5 External Influences**

11781 **4.57.5.1 Standard Input**

11782 None.

11783 **4.57.5.2 Input Files**

11784 None.

11785 **4.57.5.3 Environment Variables**

11786 The following environment variables shall affect the execution of `sleep`:

11787	LANG	This variable shall determine the locale to use for the
11788		locale categories when both LC_ALL and the correspond-
11789		ing environment variable (beginning with LC_) do not
11790		specify a locale. See 2.6.
11791	LC_ALL	This variable shall determine the locale to be used to over-
11792		ride any values for locale categories specified by the set-
11793		tings of LANG or any environment variables beginning
11794		with LC_ .

11795 **LC_CTYPE** This variable shall determine the locale for the interpreta-
 11796 tion of sequences of bytes of text data as characters (e.g.,
 11797 single- versus multibyte characters in arguments).

11798 **LC_MESSAGES** This variable shall determine the language in which mes-
 11799 sages should be written.

11800 **4.57.5.4 Asynchronous Events**

11801 If the `sleep` utility receives a `SIGALRM` signal, one of the following actions shall
 11802 be taken:

- 11803 (1) Terminate normally with a zero exit status
- 11804 (2) Effectively ignore the signal
- 11805 (3) Provide the default behavior for signals described in 2.11.5.4. This could
 11806 include terminating with a nonzero exit status.

11807 The `sleep` utility shall take the standard action for all other signals; see 2.11.5.4.

11808 **4.57.6 External Effects**

11809 **4.57.6.1 Standard Output**

11810 None.

11811 **4.57.6.2 Standard Error**

11812 Used only for diagnostic messages.

11813 **4.57.6.3 Output Files**

11814 None.

11815 **4.57.7 Extended Description**

11816 None.

11817 **4.57.8 Exit Status**

11818 The `sleep` utility shall exit with one of the following values:

- 11819 0 The execution was successfully suspended for at least *time* seconds, or a
 11820 SIGALRM signal was received (see 4.57.5.4).
- 11821 >0 An error occurred.

11822 4.57.9 Consequences of Errors

11823 Default.

11824 4.57.10 Rationale. *(This subclause is not a part of P1003.2)*

11825 Examples, Usage

11826 The exit status is allowed to be zero when `sleep` is interrupted by the SIGALRM
 11827 signal, because most implementations of this utility rely on the arrival of that sig-
 11828 nal to notify them that the requested finishing time has been successfully
 11829 attained. Such implementations thus do not distinguish this situation from the
 11830 successful completion case. Other implementations are allowed to catch the sig-
 11831 nal and go back to sleep until the requested time expires or provide the normal
 11832 signal termination procedures.

11833 History of Decisions Made

11834 As with all other utilities that take integral operands and do not specify
 11835 subranges of allowed values, `sleep` is required by this standard to deal with *time*
 11836 requests of up to 2 147 483 647 seconds. This may mean that some implementa-
 11837 tions will have to make multiple calls to the underlying operating system's delay
 11838 mechanism if its argument range is less than this.

11839 4.58 `sort` — Sort, merge, or sequence check text files

11840 4.58.1 Synopsis

11841 `sort [-m] [-o output] [-bdfinru] [-t char] [-k keydef] ... [file ...]`

11842 `sort -c [-bdfinru] [-t char] [-k keydef] ... [file]`

11843 *Obsolescent Versions:*

11844 `sort [-mu] [-o output] [-bdfinr] [-t char] [+pos1[-pos2]] ... [file ...]`

11845 `sort -c [-u] [-bdfinr] [-t char] [+pos1[-pos2]] ... [file]`

11846 4.58.2 Description

11847 The `sort` utility shall perform one of the following functions:

- 11848 (1) Sort lines of all the named files together and write the result to the
 11849 specified output.
- 11850 (2) Merge lines of all the named (presorted) files together and write the
 11851 result to the specified output.

11852 (3) Check that a single input file is correctly presorted.

11853 Comparisons shall be based on one or more sort keys extracted from each line of
 11854 input (or the entire line if no sort keys are specified), and shall be performed
 11855 using the collating sequence of the current locale.

11856 4.58.3 Options

11857 The `sort` utility shall conform to the utility argument syntax guidelines
 11858 described in 2.10.2, except that the notation `+pos1 -pos2` uses a nonstandard
 11859 prefix and multidigit option names in the obsolescent versions, the `-o output`
 11860 option shall be recognized after a *file* operand as an obsolescent feature in both
 11861 versions where the `-c` option is not specified, and the `-k keydef` option should fol-
 11862 low the `-b`, `-d`, `-f`, `-i`, `-n`, and `-r` options.

11863 The following options shall be supported by the implementation:

11864	<code>-c</code>	Check that the single input file is ordered as specified by the arguments and the collating sequence of the current locale. No output shall be produced; only the exit code shall be affected.
11865		
11866		
11867	<code>-m</code>	Merge only; the input files shall be assumed to be already sorted.
11868	<code>-o output</code>	Specify the name of an output file to be used instead of the standard output. This file can be the same as one of the input <i>files</i> .
11869		
11870	<code>-u</code>	Unique: suppress all but one in each set of lines having equal keys. If used with the <code>-c</code> option, check that there are no lines with duplicate keys, in addition to checking that the input file is sorted.
11871		
11872		
11873		

11874 The following options shall override the default ordering rules. When ordering
 11875 options appear independent of any key field specifications, the requested field ord-
 11876 ering rules shall be applied globally to all sort keys. When attached to a specific
 11877 key (see `-k`), the specified ordering options shall override all global ordering
 11878 options for that key. In the obsolescent forms, if one or more of these options fol-
 11879 lows a `+pos1` option, it shall affect only the key field specified by that preceding
 11880 option.

11881	<code>-d</code>	Specify that only <blank>s and alphanumeric characters, according to the current setting of <code>LC_CTYPE</code> , shall be significant in comparisons. The behavior is undefined for a sort key to which <code>-i</code> or <code>-n</code> also applies.
11882		
11883		
11884		
11885	<code>-f</code>	Consider all lowercase characters that have uppercase equivalents, according to the current setting of <code>LC_CTYPE</code> , to be the uppercase equivalent for the purposes of comparison.
11886		
11887		
11888	<code>-i</code>	Ignore all characters that are nonprintable, according to the current setting of <code>LC_CTYPE</code> .
11889		

- 11890 `-n` Restrict the sort key to an initial numeric string, consisting of
 11891 optional <blank>s, optional minus sign, and zero or more digits
 11892 with an optional radix character and thousands separators (as
 11893 defined in the current locale), which shall be sorted by arithmetic
 11894 value. An empty digit string shall be treated as zero. Leading
 11895 zeros and signs on zeros shall not affect ordering.
- 11896 `-r` Reverse the sense of comparisons.
- 11897 The treatment of field separators can be altered using the options:
- 11898 `-b` Ignore leading <blank>s when determining the starting and end-
 11899 ing positions of a restricted sort key. If the `-b` option is specified
 11900 before the first `-k` option, it shall be applied to all `-k` options.
 11901 Otherwise, the `-b` option can be attached independently to each
 11902 `-k field_start` or `field_end` option-argument (see below).
- 11903 `-t char` Use *char* as the field separator character; *char* shall not be con-
 11904 sidered to be part of a field (although it can be included in a sort
 11905 key). Each occurrence of *char* shall be significant (for example,
 11906 <*char*><*char*> shall delimit an empty field). If `-t` is not specified,
 11907 <blank> characters shall be used as default field separators;
 11908 each maximal nonempty sequence of <blank> characters that fol-
 11909 lows a non-<blank> character shall be a field separator.
- 11910 Sort keys can be specified using the options:
- 11911 `-k keydef` The *keydef* argument is a restricted sort key field definition. The
 11912 format of this definition is
- 11913 `field_start[type][,field_end[type]]`
- 11914 where *field_start* and *field_end* define a key field restricted to a
 11915 portion of the line (see 4.58.7), and *type* is a modifier from the list
 11916 of characters b, d, f, i, n, r. The b modifier shall behave like the
 11917 `-b` option, but applies only to the *field_start* or *field_end* to which
 11918 it is attached. The other modifiers shall behave like the
 11919 corresponding options, but shall apply only to the key field to
 11920 which they are attached; they shall have this effect if specified
 11921 with *field_start*, *field_end*, or both. Modifiers attached to a
 11922 *field_start* or *field_end* shall override any specifications made by
 11923 the options. Implementations shall support at least nine
 11924 occurrences of the `-k` option, which shall be significant in com-
 11925 mand line order. If no `-k` option is specified, a default sort key of
 11926 the entire line shall be used.
- 11927 When there are multiple key fields, later keys shall be compared
 11928 only after all earlier keys compare equal. Except when the `-u`
 11929 option is specified, lines that otherwise compare equal shall be
 11930 ordered as if none of the options `-d`, `-f`, `-i`, `-n`, or `-k` were present
 11931 (but with `-r` still in effect, if it was specified) and with all bytes in
 11932 the lines significant to the comparison. The order in which lines
 11933 that still compare equal are written is unspecified.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

- 11934 `+pos1` (Obsolescent.) Specify the start position of a key field. See 4.58.7.
 11935 `-pos2` (Obsolescent.) Specify the end position of a key field. See 4.58.7.

11936 **4.58.4 Operands**

11937 The following operand shall be supported by the implementation:

- 11938 *file* A pathname of a file to be sorted, merged, or checked. If no *file*
 11939 operands are specified, or if a *file* operand is `-`, the standard input
 11940 shall be used.

11941 **4.58.5 External Influences**

11942 **4.58.5.1 Standard Input**

11943 The standard input shall be used only if no *file* operands are specified, or if a *file*
 11944 operand is `-`. See Input Files.

11945 **4.58.5.2 Input Files**

11946 The input files shall be text files, except that the `sort` utility shall add a `<new-`
 11947 `line>` to the end of a file ending with an incomplete last line.

11948 **4.58.5.3 Environment Variables**

11949 The following environment variables shall affect the execution of `sort`:

- | | | |
|---|--------------------|---|
| 11950
11951
11952
11953 | LANG | This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6. |
| 11954
11955
11956
11957 | LC_ALL | This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ . |
| 11958 | LC_COLLATE | This variable shall determine the locale for ordering rules. |
| 11959
11960
11961
11962
11963 | LC_CTYPE | This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files) and the behavior of character classification for the <code>-b</code> , <code>-d</code> , <code>-f</code> , <code>-i</code> , and <code>-n</code> options. |
| 11964
11965 | LC_MESSAGES | This variable shall determine the language in which messages should be written. |

11966 **LC_NUMERIC** This variable shall determine the locale for the definition
 11967 of the radix character and thousands separator for the `-n`
 11968 option.

11969 **4.58.5.4 Asynchronous Events**

11970 Default.

11971 **4.58.6 External Effects**

11972 **4.58.6.1 Standard Output**

11973 Unless the `-o` or `-c` options are in effect, the standard output shall contain the
 11974 sorted input.

11975 **4.58.6.2 Standard Error**

11976 Used only for diagnostic messages. A warning message about correcting an 2
 11977 incomplete last line of an input file may be generated, but need not affect the final 2
 11978 exit status. 2

11979 **4.58.6.3 Output Files**

11980 If the `-o` option is in effect, the sorted input shall be placed in the file *output*.

11981 **4.58.7 Extended Description**

11982 The notation

11983 `-k field_start[type][,field_end[type]]`

11984 shall define a key field that begins at *field_start* and ends at *field_end* inclusive,
 11985 unless *field_start* falls beyond the end of the line or after *field_end*, in which case
 11986 the key field shall be empty. A missing *field_end* shall mean the last character of
 11987 the line.

11988 A field comprises a maximal sequence of nonseparating characters and, in the 1
 11989 absence of option `-t`, any preceding field separator. 1

11990 The *field_start* portion of the *keydef* option argument shall have the form:

11991 `field_number[.first_character]`

11992 Fields and characters within fields shall be numbered starting with 1. The
 11993 *field_number* and *first_character* pieces, interpreted as positive decimal integers,
 11994 shall specify the first character to be used as part of a sort key. If *.first_character*
 11995 is omitted, it shall refer to the first character of the field.

11996 The *field_end* portion of the *keydef* option argument shall have the form:

11997 *field_number*[*.last_character*]

11998 The *field_number* shall be as described above for *field_start*. The *last_character*
11999 piece, interpreted as a nonnegative decimal integer, shall specify the last charac-
12000 ter to be used as part of the sort key. If *last_character* evaluates to zero or
12001 *.last_character* is omitted, it shall refer to the last character of the field specified
12002 by *field_number*.

12003 If the *-b* option or *b* type modifier is in effect, characters within a field shall be
12004 counted from the first non-*<blank>* in the field. (This shall apply separately to
12005 *first_character* and *last_character*.)

12006 The obsolescent [*+pos1* [*-pos2*]] options provide functionality equivalent to the
12007 *-k keydef* option. For comparison, the full formats of these options shall be:

12008 *+field0_number*[*.first0_character*][*type*] [*-field0_number*[*.first0_character*][*type*]]
12009 *-k field_number*[*.first_character*][*type*][*,field_number*[*.last_character*][*type*]]

12010 In the obsolescent form, fields (specified by *field0_number*) and characters within
12011 fields (specified by *first0_character*) shall be numbered from zero instead of one.
12012 The *-pos2* option shall specify the first character after the sort field instead of the
12013 last character in the sort field. (Therefore, *field0_number* and *first0_character*
12014 shall be interpreted as nonnegative, instead of positive, decimal integers and
12015 there is no need for a specification of a *last_character*-like form.) The optional
12016 type modifiers shall be the same in both forms. If *.first0_character* is omitted or
12017 *first0_character* evaluates to zero, it shall refer to the first character of the field.

12018 Thus, a the fully specified *+pos1 -pos2* form:

12019 *+w.x -y.z*

12020 shall be equivalent to:

12021 *-k w+1.x+1,y.0* (if *z* == 0)
12022 *-k w+1.x+1,y+1.z* (if *z* > 0)

12023 As with the nonobsolescent forms, implementations shall support at least nine
12024 occurrences of the *+pos1* option, which shall be significant in command line order.

12025 4.58.8 Exit Status

12026 The *sort* utility shall exit with one of the following values:

- | | | |
|-------|----|--|
| 12027 | 0 | All input files were output successfully, or <i>-c</i> was specified and the |
| 12028 | | input file was correctly sorted. |
| 12029 | 1 | Under the <i>-c</i> option, the file was not ordered as specified, or if the <i>-c</i> |
| 12030 | | and <i>-u</i> options were both specified, two input lines were found with |
| 12031 | | equal keys. This exit status shall not be returned if the <i>-c</i> option is not |
| 12032 | | used. |
| 12033 | >1 | An error occurred. |

12034 **4.58.9 Consequences of Errors**

12035 Default.

12036 **4.58.10 Rationale.** *(This subclause is not a part of P1003.2)*12037 **Examples, Usage**

12038 In the following examples, nonobsolescent and obsolescent ways of specifying sort
 12039 keys are given as an aid to understanding the relationship between the two forms.

12040 Either of the following commands sorts the contents of `infile` with the second
 12041 field as the sort key:

```
12042      sort -k 2,2 infile
12043      sort +1 -2 infile
```

12044 Either of the following commands sorts, in reverse order, the contents of `infile1`
 12045 and `infile2`, placing the output in `outfile` and using the second character of
 12046 the second field as the sort key (assuming that the first character of the second
 12047 field is the field separator):

```
12048      sort -r -o outfile -k 2.2,2.2 infile1 infile2
12049      sort -r -o outfile +1.1 -1.2 infile1 infile2
```

1

12050 Either of the following commands sorts the contents of `infile1` and `infile2`
 12051 using the second non-`<blank>` character of the second field as the sort key:

```
12052      sort -k 2.2b,2.2b infile1 infile2
12053      sort +1.1b -1.2b  infile1 infile2
```

12054 Either of the following commands prints the System V password file (user data-
 12055 base) sorted by the numeric user ID (the third colon-separated field):

```
12056      sort -t : -k 3,3n /etc/passwd
12057      sort -t : +2 -3n /etc/passwd
```

12058 Either of the following commands prints the lines of the already sorted file
 12059 `infile`, suppressing all but one occurrence of lines having the same third field:

```
12060      sort -um -k 3.1,3.0 infile
12061      sort -um +2.0 -3.0 infile
```

12062 Examples in some historical documentation state that options `-um` with one input
 12063 file keep the first in each set of lines with equal keys. This behavior was deemed
 12064 to be an implementation artifact and was not made standard.

1

2

1

12065 The default value for `-t`, `<blank>`, has different properties than, for example, `-t`
 12066 "`<space>`". If a line contains:

```
12067      <space><space>foo
```

12068 the following treatment would occur with default separation versus specifically
 12069 selecting a `<space>`:

	Field	Default	-t "<space>"	
12070				
12071	1	<space><space>foo	<i>empty</i>	
12072	2	<i>empty</i>	<i>empty</i>	1
12073	3	<i>empty</i>	foo	1
12074	The leading field separator itself is included in a field when -t is not used. For			1
12075	example, this command returns an exit status of zero, meaning the input was			1
12076	already sorted:			1
12077	sort -c -k 2 <<eof			1
12078	y<tab>b			1
12079	x<space>a			1
12080	eof			1
12081	(assuming that <tab> precedes <space> in the current collating sequence). The			1
12082	field separator is not included in a field when it is explicitly set via -t. This is			1
12083	historical practice and allows usage such as			1
12084	sort -t " " -k 2n <<eof			1
12085	Atlanta 425022 Georgia			1
12086	Birmingham 284413 Alabama			1
12087	Columbia 100385 South Carolina			1
12088	eof			1
12089	where the second field can be correctly sorted numerically without regard to the			1
12090	nonnumeric field separator.			1
12091	History of Decisions Made			
12092	The -z option was removed; it is not standard practice on most systems, and is			
12093	inconsistent with using sort to individually sort several files and then merging			
12094	them together. The previous language appeared to require implementations to			
12095	determine the proper buffer length during the sort phase of operation, but not			
12096	during the merge.			
12097	The -y option was removed because of nonportability. The -M option, present in			
12098	System V, was removed because of nonportability in international usage.			
12099	An undocumented -T option exists in some implementations. It is used to specify			
12100	a directory for intermediate files. Implementations are encouraged to support the			
12101	use of the TMPDIR environment variable instead of adding an option to support			
12102	this functionality.			
12103	The -k option was added to satisfy two complaints. First, the zero-based counting			
12104	used by sort is not consistent with other utility conventions. Second, it did not			
12105	meet syntax guideline requirements. The one-based counting in this standard			
12106	was developed from the input provided by several ballot comments, ballot objec-			
12107	tions, and discussions with users.			
12108	The wording in Draft 10 also clarifies that the -b, -d, -f, -i, -n, and -r options			
12109	have to come before the first sort key specified if they are intended to apply to all			
12110	specified keys. The way it is described in this standard matches historical prac-			
12111	tice, not historical documentation. In the nonobsolescent versions, the results are			

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

12112 unspecified if these options are specified after a `-k` option. This will allow imple-
 12113 mentations to make the options independent of each other when the obsolescent
 12114 forms are finally dropped (if that ever happens).

12115 Historical documentation indicates that “setting `-n` implies `-b`.” The description
 12116 of `-n` already states that optional leading `<blank>s` are tolerated in doing the
 12117 comparison. If `-b` is enabled, rather than implied, by `-n`, this has unusual side
 12118 effects. When a character offset is used into a column of numbers (e.g., to sort
 12119 mod 100), that offset will be measured relative to the most significant digit, not to
 12120 the column. Based upon a recommendation of the author of the original `sort`
 12121 utility, the `-b` implication has been omitted from POSIX.2 and an application
 12122 wishing to achieve the previously mentioned side effects will have to manually
 12123 code the `-b` flag.

12124 **4.59 `stty` — Set the options for a terminal**

12125 **4.59.1 Synopsis**

12126 `stty` [`-a` | `-g`]

12127 `stty` *operands*

12128 **4.59.2 Description**

12129 The `stty` utility shall set or report on terminal I/O characteristics for the device
 12130 that is its standard input. Without options or operands specified, it shall report
 12131 the settings of certain characteristics, usually those that differ from
 12132 implementation-defined defaults. Otherwise, it shall modify the terminal state
 12133 according to the specified operands. Detailed information about the modes listed
 12134 in the first five groups below are described in POSIX.1 {8} Section 7. Operands in
 12135 the Combination Modes group (see 4.59.4.6) shall be implemented using operands
 12136 in the previous groups. Some combinations of operands are mutually exclusive on
 12137 some terminal types; the results of using such combinations are unspecified.

12138 Typical implementations of this utility require a communications line configured
 12139 to use a POSIX.1 {8} *termios* interface. On systems where none of these lines are
 12140 available, and on lines not currently configured to support the POSIX.1 {8} *termios*
 12141 interface, some of the operands need not affect terminal characteristics.

12142 **4.59.3 Options**

12143 The `stty` utility shall conform to the utility argument syntax guidelines
 12144 described in 2.10.2.

12145 The following options shall be supported by the implementation:

12146 -a Write to standard output all the current settings for the terminal.
 12147 -g Write to standard output all the current settings in an
 12148 unspecified form that can be used as arguments to another invo-
 12149 cation of the `stty` utility on the same system. The form used
 12150 shall not contain any characters that would require quoting to
 12151 avoid word expansion by the shell; see 3.6.

12152 **4.59.4 Operands**

12153 The following operands shall be supported by the implementation to set the termi-
 12154 nal characteristics:

12155 **4.59.4.1 Control Modes**

12156 parenb (-parenb) Enable (disable) parity generation and detection.
 12157 This shall have the effect of setting (not setting)
 12158 PARENB in the *termios* *c_cflag* field, as defined in
 12159 POSIX.1 {8}.

12160 parodd (-parodd) Select odd (even) parity. This shall have the effect of
 12161 setting (not setting) PARODD in the *termios* *c_cflag*
 12162 field, as defined in POSIX.1 {8}.

12163 cs5 cs6 cs7 cs8 Select character size, if possible. This shall have the
 12164 effect of setting CS5, CS6, CS7, and CS8, respectively,
 12165 in the *termios* *c_cflag* field, as defined in POSIX.1 {8}.

12166 *number* Set terminal baud rate to the number given, if possi-
 12167 ble. If the baud rate is set to zero, the modem control
 12168 lines shall no longer be asserted. This shall have the
 12169 effect of setting the input and output *termios* baud
 12170 rate values as defined in POSIX.1 {8}.

12171 ispeed *number* Set terminal input baud rate to the number given, if
 12172 possible. If the input baud rate is set to zero, the
 12173 input baud rate shall be specified by the value of the
 12174 output baud rate. This shall have the effect of setting
 12175 the input *termios* baud rate values as defined in
 12176 POSIX.1 {8}.

12177 ospeed *number* Set terminal output baud rate to the number given, if
 12178 possible. If the output baud rate is set to zero, the
 12179 modem control lines shall no longer be asserted. This
 12180 shall have the effect of setting the output *termios*
 12181 baud rate values as defined in POSIX.1 {8}.

12182 hupcl (-hupcl) Stop asserting modem control lines (do not stop
 12183 asserting modem control lines) on last close. This
 12184 shall have the effect of setting (not setting) HUPCL in
 12185 the *termios* *c_cflag* field, as defined in POSIX.1 {8}.

12186	hup (-hup)	Same as hupcl (-hupcl).
12187	cstopb (-cstopb)	Use two (one) stop bits per character. This shall have the effect of setting (not setting) CSTOPB in the <i>termios</i> <i>c_cflag</i> field, as defined in POSIX.1 {8}.
12188		
12189		
12190	cread (-cread)	Enable (disable) the receiver. This shall have the effect of setting (not setting) CREAD in the <i>termios</i> <i>c_cflag</i> field, as defined in POSIX.1 {8}.
12191		
12192		
12193	clocal (-clocal)	Assume a line without (with) modem control. This shall have the effect of setting (not setting) CLOCAL in the <i>termios</i> <i>c_cflag</i> field, as defined in POSIX.1 {8}.
12194		
12195		
12196	It is unspecified whether stty shall report an error if an attempt to set a Control	
12197	Mode fails.	

12198 4.59.4.2 Input Modes

12199	<code>ignbrk (-ignbrk)</code>	Ignore (do not ignore) break on input. This shall have the effect of setting (not setting) IGNBRK in the <i>termios</i> <i>c_iflag</i> field, as defined in POSIX.1 {8}.
12200		
12201		
12202	<code>brkint (-brkint)</code>	Signal (do not signal) INTR on break. This shall have the effect of setting (not setting) BRKINT in the <i>termios</i> <i>c_iflag</i> field, as defined in POSIX.1 {8}.
12203		
12204		
12205	<code>ignpar (-ignpar)</code>	Ignore (do not ignore) bytes with parity errors. This shall have the effect of setting (not setting) IGNPAR in the <i>termios</i> <i>c_iflag</i> field, as defined in POSIX.1 {8}.
12206		
12207		
12208	<code>parmrk (-parmrk)</code>	Mark (do not mark) parity errors. This shall have the effect of setting (not setting) PARMRK in the <i>termios</i> <i>c_iflag</i> field, as defined in POSIX.1 {8}.
12209		
12210		
12211	<code>inpck (-inpck)</code>	Enable (disable) input parity checking. This shall have the effect of setting (not setting) INPCK in the <i>termios</i> <i>c_iflag</i> field, as defined in POSIX.1 {8}.
12212		
12213		
12214	<code>istrip (-istrip)</code>	Strip (do not strip) input characters to seven bits. This shall have the effect of setting (not setting) ISTRIP in the <i>termios</i> <i>c_iflag</i> field, as defined in POSIX.1 {8}.
12215		
12216		
12217		
12218	<code>inlcr (-inlcr)</code>	Map (do not map) NL to CR on input. This shall have the effect of setting (not setting) INLCR in the <i>termios</i> <i>c_iflag</i> field, as defined in POSIX.1 {8}.
12219		
12220		
12221	<code>igncr (-igncr)</code>	Ignore (do not ignore) CR on input. This shall have the effect of setting (not setting) IGNCR in the <i>termios</i> <i>c_iflag</i> field, as defined in POSIX.1 {8}.
12222		
12223		
12224	<code>icrnl (-icrnl)</code>	Map (do not map) CR to NL on input. This shall have the effect of setting (not setting) ICRNL in the <i>termios</i>
12225		

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

12226		<i>c_iflag</i> field, as defined in POSIX.1 {8}.
12227	ixon (-ixon)	Enable (disable) START/STOP output control. Output from the system is stopped when the system receives STOP and started when the system receives START. This shall have the effect of setting (not setting) IXON in the <i>termios c_iflag</i> field, as defined in POSIX.1 {8}.
12228		
12229		
12230		
12231		
12232	ixoff (-ixoff)	Request that the system send (not send) STOP characters when the input queue is nearly full and START characters to resume data transmission. This shall have the effect of setting (not setting) IXOFF in the <i>termios c_iflag</i> field, as defined in POSIX.1 {8}.
12233		
12234		
12235		
12236		
12237	4.59.4.3 Output Modes	
12238	opost (-opost)	Post-process output (do not post-process output; ignore all other output modes). This shall have the effect of setting (not setting) OPOST in the <i>termios c_oflag</i> field, as defined in POSIX.1 {8}.
12239		
12240		
12241		
12242	4.59.4.4 Local Modes	
12243	isig (-isig)	Enable (disable) the checking of characters against the special control characters INTR, QUIT, and SUSP. This shall have the effect of setting (not setting) ISIG in the <i>termios c_lflag</i> field, as defined in POSIX.1 {8}.
12244		
12245		
12246		
12247	icanon (-icanon)	Enable (disable) canonical input (ERASE and KILL processing). This shall have the effect of setting (not setting) ICANON in the <i>termios c_lflag</i> field, as defined in POSIX.1 {8}.
12248		
12249		
12250		
12251	iexten (-iexten)	Enable (disable) any implementation-defined special control characters not currently controlled by icanon, isig, ixon, or ixoff. This shall have the effect of setting (not setting) IEXTEN in the <i>termios c_lflag</i> field, as defined in POSIX.1 {8}.
12252		
12253		
12254		
12255		
12256	echo (-echo)	Echo back (do not echo back) every character typed. This shall have the effect of setting (not setting) ECHO in the <i>termios c_lflag</i> field, as defined in POSIX.1 {8}.
12257		
12258		
12259		
12260	echoe (-echoe)	The ERASE character shall (shall not) visually erase the last character in the current line from the display, if possible. This shall have the effect of setting (not setting) ECHOE in the <i>termios c_lflag</i> field, as defined in POSIX.1 {8}.
12261		
12262		
12263		
12264		

12265	<code>echok (-echok)</code>	Echo (do not echo) NL after KILL character. This shall have the effect of setting (not setting) ECHOK in the <i>termios</i> <i>c_lflag</i> field, as defined in POSIX.1 {8}.	
12266			
12267			
12268	<code>echonl (-echonl)</code>	Echo (do not echo) NL, even if <code>echo</code> is disabled. This shall have the effect of setting (not setting) ECHONL in the <i>termios</i> <i>c_lflag</i> field, as defined in POSIX.1 {8}.	
12269			
12270			
12271	<code>noflsh (-noflsh)</code>	Disable (enable) flush after INTR, QUIT, SUSP. This shall have the effect of setting (not setting) NOFLSH in the <i>termios</i> <i>c_lflag</i> field, as defined in POSIX.1 {8}.	
12272			
12273			
12274	<code>tostop (-tostop)</code>	Send SIGTTOU for background output. This shall have the effect of setting (not setting) TOSTOP in the <i>termios</i> <i>c_lflag</i> field, as defined in POSIX.1 {8}.	2
12275			2
12276			2
12277		NOTE: Setting TOSTOP has no effect on systems not supporting the POSIX.1 {8} job control option.	2
12278			2

12279 4.59.4.5 Special Control Character Assignments

12280	<i>control-character string</i>	
12281		Set <i>control-character</i> to <i>string</i> . If <i>control-character</i> is one of the character sequences in the first column of Table 4-9, the corresponding POSIX.1 {8} control character from the second column shall be recognized. This shall have the effect of setting the corresponding element of the <i>termios</i> <i>c_cc</i> array (see POSIX.1 {8} 7.1.2).
12282		
12283		
12284		
12285		
12286		
12287		

12288 **Table 4-9 – stty Control Character Names**

12289	<i>control-character</i>	POSIX.1 {8} Subscript	Description
12290			
12291	<code>eof</code>	VEOF	EOF character
12292	<code>eol</code>	VEOL	EOL character
12293	<code>erase</code>	VERASE	ERASE character
12294	<code>intr</code>	VINTR	INTR character
12295	<code>kill</code>	VKILL	KILL character
12296	<code>quit</code>	VQUIT	QUIT character
12297	<code>susp</code>	VSUSP	SUSP character
12298	<code>start</code>	VSTART	START character
12299	<code>stop</code>	VSTOP	STOP character
12300			

12301		If <i>string</i> is a single character, the control character shall be set to that character. If <i>string</i> is the two-character sequence "^-" or the string "undef", the control character shall be set to {_POSIX_VDISABLE}, if it is in effect for the device; if {_POSIX_VDISABLE} is
12302		
12303		
12304		
12305		

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

not in effect for the device, it shall be treated as an error. In the POSIX Locale, if *string* is a two-character sequence beginning with circumflex (^), and the second character is one of those listed in the ^c column of Table 4-10, the control character shall be set to the corresponding character value in the Value column of the table.

Table 4-10 – stty Circumflex Control Characters

	^c	Value	^c	Value	^c	Value
12316	a, A	<SOH>	l, L	<FF>	w, W	<ETB>
12317	b, B	<STX>	m, M	<CR>	x, X	<CAN>
12318	c, C	<ETX>	n, N	<SO>	y, Y	
12319	d, D	<EOT>	o, O	<SI>	z, Z	<SUB>
12320	e, E	<ENQ>	p, P	<DLE>	[<ESC>
12321	f, F	<ACK>	q, Q	<DC1>	\	<FS>
12322	g, G	<BEL>	r, R	<DC2>]	<GS>
12323	h, H	<BS>	s, S	<DC3>	^	<RS>
12324	i, I	<HT>	t, T	<DC4>	_	<US>
12325	j, J	<LF>	u, U	<NAK>	?	
12326	k, K	<VT>	v, V	<SYN>		

min *number*
time *number* Set the value of min or time to *number*. MIN and TIME are used in noncanonical mode input processing (-icanon).

4.59.4.6 Combination Modes

saved settings Set the current terminal characteristics to the saved settings produced by the -g option.

evenp or parity Enable parenb and cs7; disable parodd.

oddp Enable parenb, cs7, and parodd.

-parity, -evenp, or -oddp Disable parenb, and set cs8.

nl (-nl) Enable (disable) icrnl. In addition, -nl unsets inlcr and igncr.

ek Reset ERASE and KILL characters back to system defaults.

sane Reset all modes to some reasonable, unspecified, values.

12345 **4.59.5 External Influences**

12346 **4.59.5.1 Standard Input**

12347 Although no input is read from standard input, standard input is used to get the
12348 current terminal I/O characteristics and to set new terminal I/O characteristics.

12349 **4.59.5.2 Input Files**

12350 None.

12351 **4.59.5.3 Environment Variables**

12352 The following environment variables shall affect the execution of `stty`:

12353	LANG	This variable shall determine the locale to use for the
12354		locale categories when both LC_ALL and the correspond-
12355		ing environment variable (beginning with LC_) do not
12356		specify a locale. See 2.6.
12357	LC_ALL	This variable shall determine the locale to be used to over-
12358		ride any values for locale categories specified by the set-
12359		tings of LANG or any environment variables beginning
12360		with LC_ .
12361	LC_CTYPE	This variable shall determine the locale for the interpreta-
12362		tion of sequences of bytes of text data as characters (e.g.,
12363		single- versus multibyte characters in arguments) and
12364		which characters are in the class <code>print</code> .
12365	LC_MESSAGES	This variable shall determine the language in which mes-
12366		sages should be written.

12367 **4.59.5.4 Asynchronous Events**

12368 Default.

12369 **4.59.6 External Effects**

12370 **4.59.6.1 Standard Output**

12371 If operands are specified, no output shall be produced.

12372 If the `-g` option is specified, `stty` shall write to standard output the current set-
12373 tings in a form that can be used as arguments to another instance of `stty` on the
12374 same system.

12375 If the `-a` option is specified, all of the information as described in 4.59.4 shall be
12376 written to standard output. Unless otherwise specified, this information shall be
12377 written as `<space>`-separated tokens in an unspecified format, on one or more

12378 lines, with an unspecified number of tokens per line. Additional information may
 12379 be written.

12380 If no options or operands are specified, an unspecified subset of the information
 12381 written for the `-a` option shall be written.

12382 If speed information is written as part of the default output, or if the `-a` option is
 12383 specified and if the terminal input speed and output speed are the same, the
 12384 speed information shall be written as follows:

12385 `"speed %d baud;", <speed>`

12386 Otherwise, speeds shall be written as:

12387 `"ispeed %d baud; ospeed %d baud;", <ispeed>, <ospeed>`

12388 In locales other than the POSIX Locale, the word `baud` may be changed to some-
 12389 thing more appropriate in those locales.

12390 If control characters are written as part of the default output, or if the `-a` option is
 12391 specified, control characters shall be written as:

12392 `"%s = %s;", <control-character name>, <value>`

12393 where *value* is either the character, or some visual representation of the character
 12394 if it is nonprintable, or the string `<undef>` if the character is disabled.

12395 **4.59.6.2 Standard Error**

12396 Used only for diagnostic messages.

12397 **4.59.6.3 Output Files**

12398 None.

12399 **4.59.7 Extended Description**

12400 None.

12401 **4.59.8 Exit Status**

12402 The `stty` utility shall exit with one of the following values:

12403 0 The terminal options were read or set successfully.

12404 >0 An error occurred.

12405 **4.59.9 Consequences of Errors**

12406 Default.

12407 **4.59.10 Rationale.** *(This subclause is not a part of P1003.2)*

12408 **Examples, Usage**

12409 Since POSIX.1 {8} doesn't specify any output modes, they are not specified in this
12410 standard either. Implementations are expected to provide `stty` operands
12411 corresponding to all of the output modes they support.

12412 In many ways outside the scope of POSIX.2, `stty` is primarily used to tailor the
12413 user interface of the terminal, such as selecting the preferred ERASE and KILL
12414 characters. As an application programming utility, `stty` can be used within shell
12415 scripts to alter the terminal settings for the duration of the script. The `-g` flag is
12416 designed to facilitate the saving and restoring of terminal state from the shell
12417 level. For example, a program may:

```
12418     saveterm="$(stty -g)"    # save terminal state
12419     stty (new settings)      # set new state
12420     ...                      # ...
12421     stty $saveterm           # restore terminal state
```

12422 Since the format is unspecified, the saved value is not portable across systems.

12423 Since the `-a` format is so loosely specified, scripts that save and restore terminal
12424 settings should use the `-g` option.

12425 **History of Decisions Made**

12426 The original `stty` manual page was taken directly from System V and reflected
12427 the System V terminal driver *termio*. It has been modified to correspond to the
12428 POSIX.1 {8} terminal driver *termios*.

12429 The *termios* section states that individual disabling of control characters is an
12430 option `{_POSIX_VDISABLE}`. If enabled, two conventions currently exist for speci-
12431 fying this: System V uses `"^-"`, and BSD uses `undef`. Both are accepted by
12432 POSIX.2 `stty`. The other BSD convention of using the letter `u` was rejected
12433 because it conflicts with the actual letter `u`, which is an acceptable value for a con-
12434 trol character.

12435 Early drafts did not specify the mapping of `^c` to control characters because the
12436 control characters were not specified in the POSIX Locale character set description
12437 file requirements. The control character set is now specified in 2.4.1, so the tradi-
12438 tional mapping is specified. Note that although the mapping corresponds to
12439 control-character key assignments on many terminals that use ISO/IEC 646 {1} (or
12440 ASCII) character encodings, the mapping specified here is to the control charac-
12441 ters, not their keyboard encodings.

12442 The combination options `raw` and `cooked` (`-raw`) were dropped from the standard
12443 because the exact values that should be set are not well understood or commonly

12444 agreed on. In particular, *termios* has no explicit RAW bit, and the options that
 12445 should be re-enabled (`-raw`) are not clear. General programming practice is to
 12446 save the terminal state, change the settings for the duration of the program, and
 12447 then reset the state. This is easy to do within a C program, however it is not pos-
 12448 sible for a single invocation of `stty` to restore the terminal state (`-raw`) without
 12449 knowledge of the prior settings. Using the `-g` option and two calls to `stty`, a
 12450 shell application could do this as described above. However, it is impossible to
 12451 implement this as a single option. Also, it is not clear that changing word size
 12452 and parity is appropriate. For example, requiring that cooked set `cs7` and
 12453 `parenb` would be disastrous for users working with 8-bit international character
 12454 sets. In general, these options are too ill-defined to be of any use.

12455 Since *termios* supports separate speeds for input and output, two new options
 12456 were added to specify each distinctly.

12457 The `ixany` input mode was removed from Draft 8 on the basis that it could not be
 12458 implemented on a POSIX.1 {8} system without extensions.

12459 Some historical implementations use standard input to get and set terminal
 12460 characteristics; others use standard output. Since input from a login TTY is usu-
 12461 ally restricted to the owner while output to a TTY is frequently open to the world,
 12462 using standard input provides fewer chances of accidentally (or mischievously)
 12463 altering the terminal settings of other users. Using standard input also allows
 12464 `stty -a` and `stty -g` output to be redirected for later use. Therefore, usage of
 12465 standard input is required by this standard.

12466 The `tostop` option was omitted from early drafts through an oversight. It is the 2
 12467 only option that requires job control to be effective, and thus could have gone into 2
 12468 the UPE as a modification to `stty`, but since all other terminal control features 2
 12469 are in the base standard, `tostop` was included as well. 2

12470 **4.60 tail — Copy the last part of a file**

12471 **4.60.1 Synopsis**

12472 `tail [-f] [-c number | -n number] [file]`

12473 *Obsolescent versions:*

12474 `tail -[number][c|l][f] [file]`

12475 `tail +[number][c|l][f] [file]`

12476 **4.60.2 Description**

12477 The `tail` utility shall copy its input file to the standard output beginning at a
12478 designated place.

12479 Copying shall begin at the point in the file indicated by the `-c number` or
12480 `-n number` options (or the \pm *number* portion of the argument to the obsolescent
12481 version). The option-argument *number* shall be counted in units of lines or bytes,
12482 according to the options `-n` and `-c` (or, in the obsolescent version, the appended
12483 option suffixes `l` or `c`).

12484 Tails relative to the end of the file may be saved in an internal buffer, and thus
12485 may be limited in length. Implementations shall ensure that such a buffer, if any,
12486 is no smaller than `{LINE_MAX}*10` bytes.

12487 **4.60.3 Options**

12488 The `tail` utility shall conform to the utility argument syntax guidelines
12489 described in standard described in 2.10.2, except that the obsolescent version
12490 accepts multicharacter options that can preceded by a plus sign.

12491 The following options shall be supported by the implementation in the nonob-
12492 solescent version:

12493 `-c number` The *number* option-argument shall be a decimal integer whose
12494 sign affects the location in the file, measured in bytes, to begin
12495 the copying:

12496	<u>Sign</u>	<u>Copying Starts</u>
12497	+	Relative to the beginning of the file.
12498	-	Relative to the end of the file.
12499	<i>none</i>	Relative to the end of the file.

12500 The origin for counting shall be 1; i.e., `-c +1` represents the first 1
12501 byte of the file, `-c -1` the last. 1

12502 *-f* If the input file is a regular file or if the *file* operand specifies a
 12503 FIFO, do not terminate after the last line of the input file has
 12504 been copied, but read and copy further bytes from the input file
 12505 when they become available. If no *file* operand is specified and
 12506 standard input is a pipe, the *-f* option shall be ignored. If the
 12507 input file is not a FIFO, pipe, or regular file, it is unspecified
 12508 whether or not the *-f* option shall be ignored.

12509 *-n number* This option shall be equivalent to *-c number*, except the starting
 12510 location in the file shall be measured in lines instead of bytes. 1
 12511 The origin for counting shall be 1; i.e., *-n +1* represents the first 1
 12512 line of the file, *-n -1* the last. 1

12513 In the obsolescent version, an argument beginning with a *-* or *+* can be used as a
 12514 single option. The argument *±number* with the letter *c* specified as a suffix shall
 12515 be equivalent to *-c ±number*; *±number* with the letter *l* specified as a suffix, or
 12516 with neither *c* nor *l* as a suffix, shall be equivalent to *-n ±number*. If *number* is
 12517 not specified in these forms, 10 shall be used. The letter *f* specified as a suffix
 12518 shall be equivalent to specifying the *-f* option. If the *-[number]c[f]* form is used
 12519 and neither *number* nor the *f* suffix is specified, it shall be interpreted as the
 12520 *-c number* option.

12521 In the nonobsolescent form, if neither *-c* nor *-n* is specified, *-n 10* shall be
 12522 assumed.

12523 **4.60.4 Operands**

12524 The following operand shall be supported by the implementation:

12525 *file* A pathname of an input file. If no *file* operands are specified, the
 12526 standard input shall be used.

12527 **4.60.5 External Influences**

12528 **4.60.5.1 Standard Input**

12529 The standard input shall be used only if no *file* operands are specified. See Input
 12530 Files.

12531 **4.60.5.2 Input Files**

12532 If the *-c* option is specified, the input file can contain arbitrary data; otherwise,
 12533 the input file shall be a text file.

12534 **4.60.5.3 Environment Variables**

12535 The following environment variables shall affect the execution of `tail`:

12536	LANG	This variable shall determine the locale to use for the
12537		locale categories when both LC_ALL and the correspond-
12538		ing environment variable (beginning with LC_) do not
12539		specify a locale. See 2.6.
12540	LC_ALL	This variable shall determine the locale to be used to over-
12541		ride any values for locale categories specified by the set-
12542		tings of LANG or any environment variables beginning
12543		with LC_ .
12544	LC_CTYPE	This variable shall determine the locale for the interpreta-
12545		tion of sequences of bytes of text data as characters (e.g.,
12546		single- versus multibyte characters in arguments and
12547		input files).
12548	LC_MESSAGES	This variable shall determine the language in which mes-
12549		sages should be written.

12550 **4.60.5.4 Asynchronous Events**

12551 Default.

12552 **4.60.6 External Effects**

12553 **4.60.6.1 Standard Output**

12554 The designated portion of the input file shall be written to standard output.

12555 **4.60.6.2 Standard Error**

12556 Used only for diagnostic messages.

12557 **4.60.6.3 Output Files**

12558 None.

12559 **4.60.7 Extended Description**

12560 None.

12561 4.60.8 Exit Status

12562 The `tail` utility shall exit with one of the following values:

12563 0 Successful completion.

12564 >0 An error occurred.

12565 4.60.9 Consequences of Errors

12566 Default.

12567 4.60.10 Rationale. *(This subclause is not a part of P1003.2)*

12568 Usage, Examples

12569 The nonobsolescent version of `tail` was created to allow conformance to the Util-
12570 ity Syntax Guidelines. The historical `-b` option was omitted because of the gen-
12571 eral nonportability of block-sized units of text. The `-c` option historically meant
12572 “characters,” but this standard indicates that it means “bytes.” This was selected
12573 to allow reasonable implementations when multibyte characters are possible; it
12574 was not named `-b` to avoid confusion with the historical `-b`.

12575 Note that the `-c` option should be used with caution when the input is a text file
12576 containing multibyte characters; it may produce output that does not start on a
12577 character boundary.

12578 The origin of counting both lines and bytes is 1, matching all widespread histori- 1
12579 cal implementations. 1

12580 The restriction on the internal buffer is a compromise between the historical
12581 System V implementation of 4K and the BSD 32K.

12582 The `-f` option can be used to monitor the growth of a file that is being written by
12583 some other process. For example, the command:

```
12584       tail -f fred
```

12585 prints the last ten lines of the file `fred`, followed by any lines that are appended
12586 to `fred` between the time `tail` is initiated and killed. As another example, the
12587 command:

```
12588       tail -f -c 15 fred
```

12589 prints the last 15 bytes of the file `fred`, followed by any bytes that are appended
12590 to `fred` between the time `tail` is initiated and killed.

12591 Although the input file to `tail` can be any type, the results need not be what
12592 would be expected on some character special device files or on file types not
12593 described by POSIX.1 {8}. Since the standard does not specify the block size used
12594 when doing input, `tail` need not read all of the data from devices that only per-
12595 form block transfers.

History of Decisions Made

The developers of the standard originally decided that `tail`, and its frequent companion, `head`, were useful mostly to interactive users, and not application programs. However, balloting input suggested that these utilities actually do find significant use in scripts, such as to write out portions of log files. The balloters also challenged the working group's assumption that clever use of `sed` could be an appropriate substitute for `tail`.

The `-f` option has been implemented as a loop that sleeps for one second and copies any bytes that are available. This is sufficient, but if more efficient methods of determining when new data are available are developed, implementations are encouraged to use them.

Historical documentation says that `tail` ignores the `-f` option if the input file is a pipe (pipe and FIFO on systems that support FIFOs). On BSD-based systems, this has been true; on System V-based systems, this was true when input was taken from standard input, but behaved as on other files if a FIFO was named as the *file* operand. Since the `-f` option is not useful on pipes and all historical implementations ignore `-f` if no *file* operand is specified and standard input is a pipe, POSIX.2 requires this behavior. However, since the `-f` option is useful on a FIFO, POSIX.2 also requires that if standard input is a FIFO or a FIFO is named, the `-f` option shall not be ignored. Although historical behavior does not ignore the `-f` option for other file types, this is unspecified so that implementations are allowed to ignore the `-f` option if it is known that the file cannot be extended.

An earlier draft had the synopsis line:

```
tail [ -c | -l ] [-f] [-n number] [file]
```

This was changed to the current form based on comments and objections noting that `-c` was almost never used without specifying a number and there was no need to specify `-l` if `-n number` was given.

12623 **4.61 tee — Duplicate standard input**

12624 **4.61.1 Synopsis**

12625 `tee [-ai] [file ...]`

12626 **4.61.2 Description**

12627 The `tee` utility shall copy standard input to standard output, making a copy in
12628 zero or more files. The `tee` utility shall not buffer output.

12629 The options determine if the specified files are overwritten or appended to.

12630 **4.61.3 Options**

12631 The `tee` utility shall conform to the utility argument syntax guidelines described
12632 in 2.10.2.

12633 The following options shall be supported by the implementation:

12634 `-a` Append the output to the files rather than overwriting them.

12635 `-i` Ignore the SIGINT signal.

12636 **4.61.4 Operands**

12637 The following operands shall be supported by the implementation:

12638 *file* A pathname of an output file. Implementations shall support pro-
12639 cessing of at least 13 *file* operands.

12640 **4.61.5 External Influences**

12641 **4.61.5.1 Standard Input**

12642 The standard input can be of any type.

12643 **4.61.5.2 Input Files**

12644 None.

12645 **4.61.5.3 Environment Variables**

12646 The following environment variables shall affect the execution of `tee`:

12647	LANG	This variable shall determine the locale to use for the
12648		locale categories when both LC_ALL and the correspond-
12649		ing environment variable (beginning with LC_) do not
12650		specify a locale. See 2.6.
12651	LC_ALL	This variable shall determine the locale to be used to over-
12652		ride any values for locale categories specified by the set-
12653		tings of LANG or any environment variables beginning
12654		with LC_ .
12655	LC_CTYPE	This variable shall determine the locale for the interpreta-
12656		tion of sequences of bytes of text data as characters (e.g.,
12657		single- versus multibyte characters in arguments).
12658	LC_MESSAGES	This variable shall determine the language in which mes-
12659		sages should be written.

12660 **4.61.5.4 Asynchronous Events**

12661 Default, except that if the **-i** option was specified, **SIGINT** shall be ignored.

12662 **4.61.6 External Effects**

12663 **4.61.6.1 Standard Output**

12664 The standard output shall be a copy of the standard input.

12665 **4.61.6.2 Standard Error**

12666 Used only for diagnostic messages.

12667 **4.61.6.3 Output Files**

12668 If any *file* operands are specified, the standard input shall be copied to each
12669 named file.

12670 **4.61.7 Extended Description**

12671 None.

12672 **4.61.8 Exit Status**

12673 **0** The standard input was successfully copied to all output files.

12674 **>0** An error occurred.

12675 **4.61.9 Consequences of Errors**

12676 If a write to any successfully opened *file* operand fails, writes to other successfully
 12677 opened *file* operands and standard output shall continue, but the exit status shall
 12678 be nonzero. Otherwise, the default actions specified in 2.11.9 shall apply.

12679 **4.61.10 Rationale.** *(This subclause is not a part of P1003.2)*

12680 **Examples, Usage**

12681 The `tee` utility is usually used in a pipeline, to make a copy of the output of some
 12682 utility.

12683 The *file* operand is technically optional, but `tee` is no more useful than `cat` when
 12684 none is specified.

12685 **History of Decisions Made**

12686 The buffering requirement means that `tee` is not allowed to use C Standard {7}
 12687 fully-buffered or line-buffered writes, not that `tee` has to do one-byte reads fol-
 12688 lowed by one-byte writes.

12689 It should be noted that early versions of BSD silently ignore any invalid options,
 12690 and accept a single `-` as an alternative to `-i`. They also print the message

12691 "tee: cannot access %s\n", <pathname>

12692 if unable to open a file.

12693 Historical implementations ignore write errors. This is explicitly not permitted
 12694 by this standard.

12695 Some historical implementations use `O_APPEND` when providing append mode;
 12696 others just `lseek()` to the end of file after opening the file without `O_APPEND`.
 12697 This standard requires functionality equivalent to using `O_APPEND`; see 2.9.1.4.

12698 **4.62 test — Evaluate expression**

12699 **4.62.1 Synopsis**

12700 `test` [*expression*]

12701 [[*expression*]]

12702 **4.62.2 Description**

12703 The `test` utility shall evaluate the *expression* and indicate the result of the 1
 12704 evaluation by its exit status. An exit status of zero indicates that the expression 1
 12705 evaluated as true and an exit status of 1 indicates that the expression evaluated 1
 12706 as false. 1

12707 In the second form of the utility, which uses [], rather than `test`, the square
 12708 brackets shall be separate arguments.

12709 **4.62.3 Options**

12710 The `test` utility shall not recognize the `--` argument in the manner specified by
 12711 utility syntax guideline 10 in 2.10.2.

12712 Implementations shall not support any options. 1

12713 **4.62.4 Operands**

12714 All operators and elements of primaries shall be presented as separate arguments 2
 12715 to the `test` utility.

12716 The following primaries can be used to construct *expression*:

- 12717 `-b file` True if *file* exists and is a block special file.
- 12718 `-c file` True if *file* exists and is a character special file.
- 12719 `-d file` True if *file* exists and is a directory.
- 12720 `-e file` True if *file* exists.
- 12721 `-f file` True if *file* exists and is a regular file.
- 12722 `-g file` True if *file* exists and its set group ID flag is set.
- 12723 `-n string` True if the length of *string* is nonzero.
- 12724 `-p file` True if *file* is a named pipe (FIFO).
- 12725 `-r file` True if *file* exists and is readable.

12726	<code>-s file</code>	True if <i>file</i> exists and has a size greater than zero.	
12727	<code>-t file_descriptor</code>		
12728		True if the file whose file descriptor number is <i>file_descriptor</i> is open and is associated with a terminal.	
12729			
12730	<code>-u file</code>	True if <i>file</i> exists and its set-user-ID flag is set.	
12731	<code>-w file</code>	True if <i>file</i> exists and is writable. True shall indicate only that the write flag is on. The <i>file</i> shall not be writable on a read-only file system even if this test indicates true.	
12732			
12733			
12734	<code>-x file</code>	True if <i>file</i> exists and is executable. True shall indicate only that the execute flag is on. If <i>file</i> is a directory, true indicates that <i>file</i> can be searched.	
12735			
12736			
12737	<code>-z string</code>	True if the length of string <i>string</i> is zero.	
12738	<i>string</i>	True if the string <i>string</i> is not the null string.	
12739	<i>s1</i> = <i>s2</i>	True if the strings <i>s1</i> and <i>s2</i> are identical.	
12740	<i>s1</i> != <i>s2</i>	True if the strings <i>s1</i> and <i>s2</i> are not identical.	
12741	<i>n1</i> -eq <i>n2</i>	True if the integers <i>n1</i> and <i>n2</i> are algebraically equal.	
12742	<i>n1</i> -ne <i>n2</i>	True if the integers <i>n1</i> and <i>n2</i> are not algebraically equal.	
12743	<i>n1</i> -gt <i>n2</i>	True if the integer <i>n1</i> is algebraically greater than the integer <i>n2</i> .	
12744			
12745	<i>n1</i> -ge <i>n2</i>	True if the integer <i>n1</i> is algebraically greater than or equal to the integer <i>n2</i> .	
12746			
12747	<i>n1</i> -lt <i>n2</i>	True if the integer <i>n1</i> is algebraically less than the integer <i>n2</i> .	
12748	<i>n1</i> -le <i>n2</i>	True if the integer <i>n1</i> is algebraically less than or equal to the integer <i>n2</i> .	
12749			
12750	A primary can be preceded by the ! operator to complement its test, as described		1
12751	below.		1
12752	The primaries with two elements of the form:		2
12753	<code>-primary_operator primary_operand</code>		2
12754	are known as <i>unary primaries</i> . The primaries with three elements in either of the		2
12755	two forms:		2
12756	<code>primary_operand -primary_operator primary_operand</code>		2
12757	<code>primary_operand primary_operator primary_operand</code>		2
12758	are known as <i>binary primaries</i> . Additional implementation-defined operators and		2
12759	<i>primary_operators</i> may be provided by implementations. They shall be of the		2
12760	form <i>-operator</i> where the first character of <i>operator</i> is not a digit. The additional		2
12761	implementation-defined operators “(” and “)” may also be provided by implemen-		2
12762	tations.		2

12763 The algorithm for determining the precedence of the operators and the return 1
 12764 value that shall be generated is based on the number of arguments presented to 1
 12765 test. (However, when using the [. . .] form, the right-bracket final argument 1
 12766 shall not be counted in this algorithm.) In the following list, \$1, \$2, \$3, and \$4 1
 12767 represent the arguments presented to test. 1

12768 0 arguments: 1

12769 Exit false (1). 1

12770 1 argument: 1

12771 Exit true (0) if \$1 is not null; otherwise, exit false. 1

12772 2 arguments: 1

12773 — If \$1 is !, exit true if \$2 is null, false if \$2 is not null. 1

12774 — If \$1 is a unary primary, exit true if the unary test is true, false if 2
 12775 the unary test is false. 1

12776 — Otherwise, produce unspecified results. 1

12777 3 arguments: 1

12778 — If \$2 is a binary primary, perform the binary test of \$1 and \$3. 2

12779 — If \$1 is !, negate the two-argument test of \$2 and \$3. 1

12780 — Otherwise, produce unspecified results. 1

12781 4 arguments: 1

12782 — If \$1 is !, negate the three-argument test of \$2, \$3, and \$4. 1

12783 — Otherwise, the results are unspecified. 1

12784 >4 arguments: 1

12785 The results are unspecified. 1

12786 4.62.5 External Influences

12787 4.62.5.1 Standard Input

12788 None.

12789 4.62.5.2 Input Files

12790 None.

12791 **4.62.5.3 Environment Variables**

12792 The following environment variables shall affect the execution of `test`:

12793	LANG	This variable shall determine the locale to use for the
12794		locale categories when both LC_ALL and the correspond-
12795		ing environment variable (beginning with LC_) do not
12796		specify a locale. See 2.6.
12797	LC_ALL	This variable shall determine the locale to be used to over-
12798		ride any values for locale categories specified by the set-
12799		tings of LANG or any environment variables beginning
12800		with LC_ .
12801	LC_CTYPE	This variable shall determine the locale for the interpreta-
12802		tion of sequences of bytes of text data as characters (e.g.,
12803		single- versus multibyte characters in arguments).
12804	LC_MESSAGES	This variable shall determine the language in which mes-
12805		sages should be written.

12806 **4.62.5.4 Asynchronous Events**

12807 Default.

12808 **4.62.6 External Effects**

12809 **4.62.6.1 Standard Output**

12810 None.

12811 **4.62.6.2 Standard Error**

12812 Used only for diagnostic messages.

12813 **4.62.6.3 Output Files**

12814 None.

12815 **4.62.7 Extended Description**

12816 None.

12817 **4.62.8 Exit Status**12818 The `test` utility shall exit with one of the following values:

- 12819 0 *expression* evaluated to true.
- 12820 1 *expression* evaluated to false or *expression* was missing.
- 12821 >1 An error occurred.

12822 **4.62.9 Consequences of Errors**

12823 Default.

12824 **4.62.10 Rationale.** (*This subclause is not a part of P1003.2*)12825 **Examples, Usage**

12826 *Editor's Note: The rationale has been rearranged quite a bit. Only new, not* 1
 12827 *moved, text has been diffmarked.* 1

12828 Historical systems have supported more than four arguments, but there has been 1
 12829 a fundamental disagreement between BSD and System V on certain combinations 1
 12830 of arguments. Since no accommodation could be reached between the two ver- 1
 12831 sions of `test` without breaking numerous applications, the version of `test` in 1
 12832 POSIX.2 specifies only the relatively simple tests and relies on the syntax of the 1
 12833 shell command language for the construction of more complex expressions. Using 1
 12834 the POSIX.2 rules produces completely reliable, portable scripts, which is not 1
 12835 always possible using either of the historical forms. Some of the historical 1
 12836 behavior is described here to aid conversion of scripts with complex `test` expres- 1
 12837 sions. 1

12838 Both BSD and System V support the combining of primaries with the following 1
 12839 constructs: 1

12840 *expression1* `-a` *expression2* True if both *expression1* and *expression2* are 1
 12841 true. 1

12842 *expression1* `-o` *expression2* True if at least one of *expression1* and *expres-* 1
 12843 sion2 are true. 1

12844 (*expression*) True if *expression* is true. 1

12845 In evaluating these more complex combined expressions, the following precedence 1
 12846 rules are used: 1

- 12847 — The unary primaries have higher precedence than the algebraic binary pri- 1
 12848 maries. 1
- 12849 — On BSD systems, the unary primaries have higher precedence than the 1
 12850 string binary primaries. On System V systems, the unary primaries have 1
 12851 lower precedence than the string binary primaries. 1

12852	— The unary and binary primaries have higher precedence than the unary	1
12853	<i>string</i> primary.	1
12854	— The <code>!</code> operator has higher precedence than the <code>-a</code> operator and the <code>-a</code>	1
12855	operator has higher precedence than the <code>-o</code> operator.	1
12856	— The <code>-a</code> and <code>-o</code> operators are left associative.	1
12857	— The parentheses can be used to alter the normal precedence and associa-	1
12858	tivity.	1
12859	The following guidance is offered for the use of the historical expressions:	1
12860	— Scripts should be careful when dealing with user-supplied input that could	1
12861	be confused with primaries and operators. Unless the application writer	
12862	knows all the cases that produce input to the script, invocations like:	
12863	<code>test "\$1" -a "\$2"</code>	
12864	should be written as:	
12865	<code>test "\$1" && test "\$2"</code>	1
12866	to avoid problems if a user-supplied values such as <code>\$1</code> set to <code>!</code> and <code>\$2</code> set to	
12867	the null string. That is, in cases where portability between implementa-	
12868	tions based on BSD and System V systems is of concern, replace:	
12869	<code>test expr1 -a expr2</code>	
12870	with:	
12871	<code>test expr1 && test expr2</code>	
12872	and replace:	
12873	<code>test expr1 -o expr2</code>	
12874	with:	
12875	<code>test expr1 test expr2</code>	
12876	but note that, in <code>test</code> , <code>-a</code> has higher precedence than <code>-o</code> while <code>&&</code> and <code> </code>	
12877	have equal precedence in the shell.	
12878	Parentheses or braces can be used in the shell command language to effect	1
12879	grouping. Historical <code>test</code> implementations also support parentheses, but	1
12880	they must be escaped when using <code>sh</code> ; for example:	1
12881	<code>test \(expr1 -a expr2 \) -o expr3</code>	1
12882	This command is not always portable. The following form can be used	1
12883	instead:	1
12884	<code>(test expr1 && test expr2) test expr3</code>	1
12885	— The two commands:	1
12886	<code>test "\$1"</code>	1
12887	<code>test ! "\$1"</code>	1
12888	could not be used reliably on historical systems. Unexpected results would	1

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

12889 occur if such a *string* expression were used and \$1 expanded to !, (, or a 1
 12890 known unary primary. Better constructs were: 1

```
12891     test -n "$1" 1
```

```
12892     test -z "$1" 1
```

12893 respectively. These suggested replacements have always worked on histor- 1
 12894 ical BSD-based implementations, and work on historical System V-based 1
 12895 implementations as long as \$1 does not expand to = or !=. Using the 1
 12896 POSIX.2 rules, any of the four forms shown will work for any possible value 1
 12897 of \$1. 1

12898 — Historical systems were also unreliable given the common construct: 1

```
12899     test "$response" = "expected string" 1
```

12900 One of the following was a more reliable form: 1

```
12901     test "X$response" = "Xexpected string"
```

```
12902     test "expected string" = "$response"
```

12903 Note that the second form assumes that `expected string` could not be
 12904 confused with any unary primary. If `expected string` starts with -,
 12905 (, !, or even =, the first form should be used instead. Using the POSIX.2
 12906 rules, any of the three comparison forms is reliable, given any input. (How-
 12907 ever, note that the strings are quoted in all cases.)

12908 The BSD and System V versions of `-f` are not the same. The BSD definition was:

```
12909     -f file      True if file exists and is not a directory.
```

12910 The *SVID* version (true if the file exists and is a regular file) was chosen for this
 12911 standard because its use is consistent with the `-b`, `-c`, `-d`, and `-p` operands (*file*
 12912 exists and is a specific file type).

12913 The `-e` primary, possessing similar functionality to that provided by the C-shell,
 12914 was added because it provides the only way for a shell script to find out if a file
 12915 exists without trying to open the file. (Since implementations are allowed to add
 12916 additional file types, a portable script cannot use:

```
12917     test -b foo -o -c foo -o -d foo -o -f foo -o -p foo
```

12918 to find out if `foo` is an existing file.) On historical BSD systems, the existence of a
 12919 file could be determined by:

```
12920     test -f foo -o -d foo
```

12921 but there was no easy way to determine that an existing file was a regular file.
 12922 An earlier draft used the KornShell `-a` primary (with the same meaning), but this
 12923 was changed to `-e` because there were concerns about the high probability of
 12924 humans confusing the `-a` primary with the `-a` binary operator.

History of Decisions Made

The `-a` and `-o` binary operators and the grouping parentheses were omitted from POSIX.2 due to a difference between existing implementations of the `test` utility in the precedence of the binary primaries `=` and `!=` compared to the unary primaries `-b`, `-c`, `-d`, `-f`, `-g`, `-n`, `-p`, `-r`, `-s`, `-t`, `-u`, `-w`, `-x`, and `-z`. On BSD, Version 7, PWB, and 32V systems the unary primaries have higher precedence than the binary operators; on System III and System V implementations, the binary operators `=` and `!=` have higher precedence. The change was apparently made for System III so that the construct:

```
test "$1" = "$2"
```

could be made to work even if `$1` started with `-`. It is believed that this change was a mistake because:

- It is not a complete solution; if `$1` expands to `(` or `!`, it still will not work.
- It makes it impossible to use the unary primaries `-n` and `-z` to test for a null string if there is any chance that the string will expand to `=`.
- More importantly, there was the well known workaround of specifying:

```
test X"$1" = X"$2"
```

that always worked.

Unfortunately, when the `=` and `!=` binary primaries were given precedence over the unary primaries, there was no workaround provided for scripts that wanted to reliably specify something like:

```
test -n "$1"
```

because if `$1` expands to `=`, it gives a syntax error.

There was some discussion of outlawing the System V behavior and requiring the more logical precedence that originated in its predecessors and remains in BSD-based systems. However, there are simply too many historical applications that would break if System V were required to make this change; this number dwarfed the number of scripts using combination logic that would then no longer be strictly portable.

POSIX.2 requires that if `test` is called with one, two, three, or four operands it correctly interprets the expression even if there is an alternate syntax tree that could lead to a syntax error. It eliminates the requirement that many string comparisons be protected with leading characters, such as

```
test X"$1" = X"$2"
```

and allows the single-argument *string* form to be used with all possible inputs.

The following examples show some of the changes that are required to be made to make historical BSD and System V-based implementations of `test` conform to this standard:

```
test -d =          POSIX.2    True if there is a directory named =
```

12964		BSD	True if there is a directory named =	
12965		System V	Syntax error; = needs two operands	
12966	test -d = -f	POSIX.2	False	
12967		BSD	Syntax error; it expects -a or -o after -d =	
12968		System V	False	
12969	Implementations are prohibited from extending test with options because it			
12970	would make the “test string” case ambiguous for inputs that might match an			
12971	extended option. Implementations can add primaries and operators, as indicated.			
12972	The following options were not included in POSIX.2, although they are provided by			
12973	some historical implementations, since these facilities and concepts are not sup-			
12974	ported by POSIX.1 {8}, nor defined in POSIX.2. These operands should not be used			
12975	by new implementations for other purposes.			
12976	-h file	True if file exists and is a symbolic link.		
12977	-k file	True if file exists and its sticky bit is set.		
12978	-L file	True if file is a symbolic link.		
12979	-C file	True if file is a contiguous file.		
12980	-S file	True if file is a socket.		
12981	-v file	True if file is a version file.		
12982	The following option was not included because it was undocumented in most			
12983	implementations, has been removed from some implementations (including			
12984	System V), and the functionality is provided by the shell (see 3.6.2).			
12985	-l string	The length of the string string.		
12986	The -b, -c, -g, -p, -u, and -x operands are derived from the SVID; historical BSD			
12987	does not provide them. The -k operand is derived from System V; historical BSD			
12988	does not provide it.			
12989	On historical BSD systems, test -w directory always returned false because test			
12990	tried to open the directory for writing, which always fails.			
12991	Some additional primaries newly invented or from the KornShell appeared in an			
12992	earlier draft as part of the Conditional Command ([[]): s1 > s2, s1 < s2, str =			
12993	pattern, str != pattern, f1 -nt f2, f1 -ot f2, and f1 -ef f2. They were not carried			
12994	forward into the test utility when the Conditional Command was removed from			
12995	the shell because they have not been included in the test utility built into histor-			
12996	ical implementations of the sh utility.			
12997	The -t file_descriptor primary is shown with a mandatory argument because the			
12998	grammar is ambiguous if it can be omitted. Historical implementations have			
12999	allowed it to be omitted, providing a default of 1.			

13000 **4.63 touch — Change file access and modification times**

13001 **4.63.1 Synopsis**

13002 `touch [-acm] [-r ref_file | -t time] file ...`

13003 *Obsolescent Version:*

13004 `touch [-acm] [date_time] file ...`

13005 **4.63.2 Description**

13006 The `touch` utility shall change the modification and/or access times of files. The
 13007 modification time is equivalent to the value of the *st_mtime* member of the *stat*
 13008 structure for a file, as described in POSIX.1 {8}; the access time is equivalent to the
 13009 value of *st_atime*.

13010 The time used can be specified by the `-t time` option-argument, the correspond-
 13011 ing time field(s) of the file referenced by the `-r ref_file` option-argument, or the
 13012 *date_time* operand, as specified in the following subclauses. If none of these are
 13013 specified, `touch` shall use the current time [the value returned by the equivalent
 13014 of the POSIX.1 {8} *time()* function].

13015 For each *file* operand, `touch` shall perform actions equivalent to the following
 13016 functions defined in POSIX.1 {8}:

- 13017 (1) If *file* does not exist, a *creat()* function call is made with the *file* operand
 13018 used as the *path* argument and the value of the bitwise inclusive OR of
 13019 S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH used
 13020 as the *mode* argument.
- 13021 (2) The *utime()* function is called with the following arguments:
 - 13022 (a) The *file* operand is used as the *path* argument.
 - 13023 (b) The *utimbuf* structure members *actime* and *modtime* are deter-
 13024 mined as described under 4.63.3.

13025 **4.63.3 Options**

13026 The `touch` utility shall conform to the utility argument syntax guidelines
 13027 described in 2.10.2.

13028 The following options shall be supported by the implementation:

- | | | |
|-------|-----------------|--|
| 13029 | <code>-a</code> | Change the access time of <i>file</i> . Do not change the modification |
| 13030 | | time unless <code>-m</code> is also specified. |

13031	−c	Do not create a specified <i>file</i> if it does not exist. Do not write any						
13032		diagnostic messages concerning this condition.						
13033	−m	Change the modification time of <i>file</i> . Do not change the access						
13034		time unless −a is also specified.						
13035	−r <i>ref_file</i>	Use the corresponding time of the file named by the pathname						
13036		<i>ref_file</i> instead of the current time.						
13037	−t <i>time</i>	Use the specified <i>time</i> instead of the current time. The option-						
13038		argument shall be a decimal number of the form:						
13039		[[<i>CC</i>][<i>YY</i>] <i>MMDDhhmm</i> [. <i>SS</i>]						
13040		where each two digits represents the following:						
13041	<i>MM</i>	The month of the year (01-12).						
13042	<i>DD</i>	The day of the month (01-31).						
13043	<i>hh</i>	The hour of the day (00-23).						
13044	<i>mm</i>	The minute of the hour (00-59).						
13045	<i>CC</i>	The first two digits of the year (the century).						
13046	<i>YY</i>	The second two digits of the year.						
13047	<i>SS</i>	The second of the minute (00-61).						
13048		Both <i>CC</i> and <i>YY</i> shall be optional. If neither is given, the current						
13049		year shall be assumed. If <i>YY</i> is specified, but <i>CC</i> is not, <i>CC</i> shall						
13050		be derived as follows:						
13051		<table><tr><td><u>If <i>YY</i> is:</u></td><td><u><i>CC</i> becomes:</u></td></tr><tr><td>69-99</td><td>19</td></tr><tr><td>00-68</td><td>20</td></tr></table>	<u>If <i>YY</i> is:</u>	<u><i>CC</i> becomes:</u>	69-99	19	00-68	20
<u>If <i>YY</i> is:</u>	<u><i>CC</i> becomes:</u>							
69-99	19							
00-68	20							
13052								
13053								
13054		The resulting time shall be affected by the value of the TZ						
13055		environment variable. If the resulting time value precedes the						
13056		Epoch, touch shall exit immediately with an error status. The						
13057		range of valid times past the Epoch is implementation defined,						
13058		but shall extend to at least midnight 1 January 2000 UTC.						
13059		The range for <i>SS</i> is (00-61) rather than (00-59) because of leap						
13060		seconds. If <i>SS</i> is 60 or 61, and the resulting time, as affected by						
13061		the TZ environment variable, does not refer to a leap second: the						
13062		resulting time shall be one or two seconds after a time where <i>SS</i>						
13063		is 59. If <i>SS</i> is not given a value, it is assumed to be zero.						
13064	If neither the −a nor −m options were specified, touch shall behave as if both the							
13065	−a and −m options were specified.							

13066 4.63.4 Operands

13067 The following operands shall be supported by the implementation:

13068 *file* A pathname of a file whose times are to be modified.
 13069 *date_time* (Obsolescent.) Use the specified *date_time* instead of the current
 13070 time. The operand is a decimal number of the form:

13071 *MMDDhhmm*[*yy*]

13072 where *MM*, *DD*, *hh*, and *mm* are as described for the *time* option-
 13073 argument to the *-t* option and the optional *yy* is interpreted as
 13074 follows:

13075 If not specified, the current year shall be used. If *yy* is in
 13076 the range 69-99, the year 1969-1999, respectively, shall be
 13077 used. Otherwise, the results are unspecified.

13078 If no *-r* option is specified, no *-t* option is specified, at least two
 13079 operands are specified, and the first operand is an eight- or ten-
 13080 digit decimal integer, the first operand shall be assumed to be a
 13081 *date_time* operand. Otherwise, the first operand shall be
 13082 assumed to be a *file* operand.

13083 4.63.5 External Influences

13084 4.63.5.1 Standard Input

13085 None.

13086 4.63.5.2 Input Files

13087 None.

13088 4.63.5.3 Environment Variables

13089 The following environment variables shall affect the execution of *touch*:

13090 **LANG** This variable shall determine the locale to use for the
 13091 locale categories when both **LC_ALL** and the correspond-
 13092 ing environment variable (beginning with **LC_**) do not
 13093 specify a locale. See 2.6.

13094 **LC_ALL** This variable shall determine the locale to be used to over-
 13095 ride any values for locale categories specified by the set-
 13096 tings of **LANG** or any environment variables beginning
 13097 with **LC_**.

13098	LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g.,
13099		single- versus multibyte characters in arguments).
13100		
13101	LC_MESSAGES	This variable shall determine the language in which messages should be written.
13102		
13103	TZ	If the <i>time</i> option-argument (or operand; see above) is
13104		specified, TZ shall be used to interpret the time for the
13105		specified time zone.

13106 **4.63.5.4 Asynchronous Events**

13107 Default.

13108 **4.63.6 External Effects**

13109 **4.63.6.1 Standard Output**

13110 None.

13111 **4.63.6.2 Standard Error**

13112 Used only for diagnostic messages.

13113 **4.63.6.3 Output Files**

13114 None.

13115 **4.63.7 Extended Description**

13116 None.

13117 **4.63.8 Exit Status**

13118 The `touch` utility shall exit with one of the following values:

- | | | |
|-------|----|--|
| 13119 | 0 | The utility executed successfully and all requested changes were made. |
| 13120 | >0 | An error occurred. |

4.63.9 Consequences of Errors

Default.

4.63.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The functionality of `touch` is described almost entirely through references to functions in POSIX.1 {8}. In this way, there is no duplication of effort required for describing such side effects as the relationship of user IDs to the user database, permissions, etc.

The interpretation of time is taken to be “seconds since the Epoch,” as defined by 2.2.2.129. It should be noted that POSIX.1 {8} conforming implementations do not take leap seconds into account when computing seconds since the Epoch. When `SS=60` is used on POSIX.1 {8} conforming implementations, the resulting time always refers to 1 plus “seconds since the Epoch” for a time when `SS=59`.

Note that although the `-t time` option-argument and the obsolescent `date_time` operand specify values in 1969, the access time and modification time fields are defined in terms of seconds since the Epoch (midnight on 1 January 1970 UTC). Therefore, depending on the value of `TZ` when `touch` is run, there will never be more than a few valid hours in 1969 and there need not be any valid times in 1969.

History of Decisions Made

There are some significant differences between this `touch` and those in System V and BSD systems. They are upward compatible for existing applications from both implementations.

- (1) In System V, an ambiguity exists when a pathname that is a decimal number leads the operands; it is treated as a time value. In BSD, no *time* value is allowed; files may only be touched to the current time. The `[-t time]` construct solves these problems for future portable applications (note that the `-t` option is not existing practice).
- (2) The inclusion of the century digits, *CC*, is also new. Note that a ten-digit *time* value is treated as if *YY*, and not *CC*, were specified. The caveat about the range of dates following the Epoch was included as recognition that some UNIX systems will not be able to represent dates beyond the January 18, 2038, because they use *signed int* as a time holder.

One ambiguous situation occurs if `-t time` is not specified, `-r ref_file` is not specified, and the first operand is an eight- or ten-digit decimal number. A portable script can avoid this problem by using:

```
touch -- file
```

or

13159 touch ./file

13160 in this case.

13161 The `-r` option was added because several comments requested this capability.
 13162 This option was named `-f` in an earlier draft, but was changed because the `-f`
 13163 option is used in the BSD version of `touch` with a different meaning.

13164 At least one historical implementation of `touch` incremented the exit code if `-c`
 13165 was specified and the file did not exist. This standard requires exit status zero if
 13166 no errors occur.

13167 **4.64 tr — Translate characters**

13168 **4.64.1 Synopsis**

13169 `tr [-cs] string1 string2`

13170 `tr -s [-c] string1`

13171 `tr -d [-c] string1`

13172 `tr -ds [-c] string1 string2`

13173 **4.64.2 Description**

13174 The `tr` utility shall copy the standard input to the standard output with substitu-
 13175 tion or deletion of selected characters. The options specified and the *string1* and
 13176 *string2* operands shall control translations that occur while copying characters
 13177 and collating elements.

13178 **4.64.3 Options**

13179 The `tr` utility shall conform to the utility argument syntax guidelines described
 13180 in 2.10.2.

13181 The following options shall be supported by the implementation:

13182 `-c` Complement the set of characters specified by *string1*. See 4.64.7.

13183 `-d` Delete all occurrences of input characters that are specified by
 13184 *string1*.

13185 `-s` Replace instances of repeated characters with a single character, 1
 13186 as described in 4.64.7. 1

13187 4.64.4 Operands

13188 The following operands shall be supported by the implementation:

13189	<i>string1</i>	
13190	<i>string2</i>	Translation control strings. Each string shall represent a set of
13191		characters to be converted into an array of characters used for the
13192		translation. For a detailed description of how the strings are
13193		interpreted, see 4.64.7.

13194 4.64.5 External Influences

13195 4.64.5.1 Standard Input

13196 The standard input can be any type of file.

13197 4.64.5.2 Input Files

13198 None.

13199 4.64.5.3 Environment Variables

13200 The following environment variables shall affect the execution of `tr`:

13201	LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
13202		
13203		
13204		
13205	LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
13206		
13207		
13208		
13209	LC_COLLATE	This variable shall determine the behavior of range expressions and equivalence classes.
13210		
13211	LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments) and the behavior of character classes.
13212		
13213		
13214		
13215	LC_MESSAGES	This variable shall determine the language in which messages should be written.
13216		

13217 4.64.5.4 Asynchronous Events

13218 Default.

13219 4.64.6 External Effects

13220 4.64.6.1 Standard Output

13221 The `tr` output shall be identical to the input, with the exception of the specified
13222 transformations.

13223 4.64.6.2 Standard Error

13224 Used only for diagnostic messages.

13225 4.64.6.3 Output Files

13226 None.

13227 4.64.7 Extended Description

13228 The operands *string1* and *string2* (if specified) define two arrays of characters or
13229 collating elements. The following conventions can be used to specify characters or
13230 collating elements:

13231	<i>character</i>	Any character not described by one of the conventions below	
13232		shall represent itself.	
13233	<i>\octal</i>	Octal sequences can be used to represent characters with	
13234		specific coded values. An octal sequence shall consist of a	
13235		backslash followed by the longest sequence of one-, two-, or	
13236		three-octal-digit characters (01234567). The sequence shall	
13237		cause the character whose encoding is represented by the one-,	
13238		two-, or three-digit octal integer to be placed into the array. If	1
13239		the size of a byte on the system is greater than nine bits, the	1
13240		valid escape sequence used to represent a byte is	1
13241		implementation-defined. Multibyte characters require multi-	1
13242		ple, concatenated escape sequences of this type, including the	1
13243		leading <code>\</code> for each byte.	1
13244	<i>\character</i>	The backslash-escape sequences in Table 2-15 (see 2.12) shall	
13245		be supported. The results of using any other character, other	
13246		than an octal digit, following the backslash are unspecified.	
13247	<i>c-c</i>	Represents the range of collating elements between the range	2
13248		endpoints, inclusive, as defined by the current setting of the	
13249		LC_COLLATE locale category. The starting endpoint shall pre-	
13250		cede the second endpoint in the current collation order. The	
13251		characters or collating elements in the range shall be placed in	
13252		the array in ascending collation sequence. No multicharacter	
13253		collating elements shall be included in the range.	
13254	<i>[:class:]</i>	Represents all characters belonging to the defined character	
13255		class, as defined by the current setting of the LC_CTYPE locale	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

category. The following character class names shall be accepted when specified in *string1*:

13258	alnum	cntrl	lower	space
13259	alpha	digit	print	upper
13260	blank	graph	punct	xdigit

When the `-d` and `-s` options are specified together, any of the character class names shall be accepted in *string2*. Otherwise, only character class names `lower` or `upper` shall be accepted in *string2* and then only if the corresponding character class (`upper` and `lower`, respectively) is specified in the same relative position in *string1*. Such a specification shall be interpreted as a request for case conversion. When `[:lower:]` appears in *string1* and `[:upper:]` appears in *string2*, the arrays shall contain the characters from the `toupper` mapping in the `LC_CTYPE` category of the current locale. When `[:upper:]` appears in *string1* and `[:lower:]` appears in *string2*, the arrays shall contain the characters from the `tolower` mapping in the `LC_CTYPE` category of the current locale. The first character from each mapping pair shall be in the array for *string1* and the second character from each mapping pair shall be in the array for *string2* in the same relative position.

Except for case conversion, the characters specified by a character class expression shall be placed in the array in an unspecified order.

If the name specified for *class* does not define a valid character class in the current locale, the behavior is undefined.

[*=equiv*=] Represents all characters or collating elements belonging to the same equivalence class as *equiv*, as defined by the current setting of the `LC_COLLATE` locale category. An equivalence class expression shall be allowed only in *string1*, or in *string2* when it is being used by the combined `-d` and `-s` options. The characters belonging to the equivalence class shall be placed in the array in an unspecified order.

[*x*n*] Represents *n* repeated occurrences of the character or collating symbol *x*. Because this expression is used to map multiple characters to one, it is only valid when it occurs in *string2*. If *n* is omitted or is zero, it shall be interpreted as large enough to extend the *string2*-based sequence to the length of the *string1*-based sequence. If *n* has a leading zero, it shall be interpreted as an octal value. Otherwise, it shall be interpreted as a decimal value.

13297 When the `-d` option is not specified:

- 13298 — Each input character or collating element found in the array specified by
13299 *string1* shall be replaced by the character or collating element in the same
13300 relative position in the array specified by *string2*. When the array specified
13301 by *string2* is shorter than the one specified by *string1*, the results are
13302 unspecified.
- 13303 — If the `-c` option is specified without `-d`, the complement of the characters
13304 specified by *string1*—the set of all characters in the current character set,
13305 as defined by the current setting of `LC_CTYPE`, except for those actually
13306 specified in the *string1* operand—shall be placed in the array in ascending
13307 collation sequence, as defined by the current setting of `LC_COLLATE`.
- 13308 — Because the order in which characters specified by character class expres-
13309 sions or equivalence class expressions is undefined, such expressions
13310 should only be used if the intent is to map several characters into one. An
13311 exception is case conversion, as described previously.

13312 When the `-d` option is specified:

- 13313 — Input characters or collating elements found in the array specified by
13314 *string1* shall be deleted.
- 13315 — When the `-c` option is specified with `-d`, all characters except those
13316 specified by *string1* shall be deleted. The contents of *string2* shall be
13317 ignored, unless the `-s` option is also specified.
- 13318 — The same string cannot be used for both the `-d` and the `-s` option; when
13319 both options are specified, both *string1* (used for deletion) and *string2* (used
13320 for squeezing) shall be required.

13321 When the `-s` option is specified, after any deletions or translations have taken
13322 place, repeated sequences of the same character shall be replaced by one
13323 occurrence of the same character, if the character is found in the array specified
13324 by the last operand. If the last operand contains a character class, such as the fol-
13325 lowing example:

```
13326 tr -s '[:space:]'
```

13327 the last operand's array shall contain all of the characters in that character class.
13328 However, in a case conversion, as described previously, such as

```
13329 tr -s '[:upper:]' '[:lower:]'
```

13330 the last operand's array shall contain only those characters defined as the second
13331 characters in each of the `toupper` or `tolower` character pairs, as appropriate.

13332 4.64.8 Exit Status

13333 The `tr` utility shall exit with one of the following values:

13334 0 All input was processed successfully.

13335 >0 An error occurred.

13336 4.64.9 Consequences of Errors

13337 Default.

13338 4.64.10 Rationale. *(This subclause is not a part of P1003.2)*

13339 Examples, Usage

13340 If necessary, *string1* and *string2* can be quoted to avoid pattern matching by the
13341 shell.

13342 The following example creates a list of all words in *file1* one per line in *file2*,
13343 where a word is taken to be a maximal string of letters.

```
13344      tr -cs "[:alpha:]" "[\n*]" <file1 >file2
```

13345 If an ordinary digit (representing itself) is to follow an octal sequence, the octal
13346 sequence must use the full three digits to avoid ambiguity.

13347 When *string2* is shorter than *string1*, a difference results between historical
13348 System V and BSD systems. A BSD system will pad *string2* with the last charac-
13349 ter found in *string2*. Thus, it is possible to do the following:

```
13350      tr 0123456789 d
```

13351 which would translate all digits to the letter d. Since this area is specifically
13352 unspecified in the standard, both the BSD and System V behaviors are allowed,
13353 but a conforming application cannot rely on the BSD behavior. It would have to
13354 code the example in the following way:

```
13355      tr 0123456789 '[d*]'
```

13356 It should be noted that, despite similarities in appearance, the string operands
13357 used by `tr` are not regular expressions.

13358 On historical System V systems, a range expression requires enclosing square- 2
13359 brackets, such as: 2

```
13360      tr '[a-z]' '[A-Z]' 2
```

13361 However, BSD-based systems did not require the brackets and this convention is 2
13362 used by POSIX.2 to avoid breaking large numbers of BSD scripts: 2

```
13363      tr a-z A-Z 2
```

13364 The preceding System V script will continue to work because the brackets, treated 2
13365 as regular characters, are translated to themselves. However, any System V 2

13366 script that relied on a-z representing the three characters a, -, and z will have to 2
 13367 be rewritten as az- or a\ -z. 2

13368 **History of Decisions Made**

13369 In some earlier drafts, an explicit option, -n, was added to disable the historical
 13370 behavior of stripping NUL characters from the input. It was felt that automati-
 13371 cally stripping NUL characters from the input was not correct functionality. How-
 13372 ever, the removal of -n in a later draft does not remove the requirement that tr
 13373 correctly process NUL characters in its input stream. NUL characters can be
 13374 stripped by using tr -d '\000'.

13375 Historical implementations of tr differ widely in syntax and behavior. For exam-
 13376 ple, the BSD version has not needed the bracket characters for the repetition
 13377 sequence. The POSIX.2 tr syntax is based more closely on the System V and
 13378 XPG3 model, while attempting to accommodate historical BSD implementations.
 13379 In the case of the short *string2* padding, the decision was to unspecify the
 13380 behavior and preserve System V and XPG scripts, which might find difficulty with
 13381 the BSD method. The assumption was made that BSD users of tr will have to
 13382 make accommodations to meet the POSIX.2 syntax anyway, and since it is possible
 13383 to use the repetition sequence to duplicate the desired behavior, whereas there is
 13384 no simple way to achieve the System V method, this was the correct, if not desir-
 13385 able, approach.

13386 The use of octal values to specify control characters, while having historical pre-
 13387 cedents, is not portable. The introduction of escape sequences for control charac-
 13388 ters should provide the necessary portability. It is recognized that this may cause
 13389 some historical scripts to break.

13390 A previous draft included support for multicharacter collating elements. Several
 13391 balloters pointed out that, while tr does employ some syntactical elements from
 13392 regular expressions, the aim of tr is quite different; ranges, for instance, do not
 13393 mean the same thing (“any of the chars in the range matches,” versus “translate
 13394 each character in the range to the output counterpart”). As a result, the previ-
 13395 ously included support for multicharacter collating elements has been removed.
 13396 What remains are ranges in current collation order (to support, e.g., accented
 13397 characters), character classes, and equivalence classes.

13398 In XPG3, the [:class:] and [=equiv=] conventions are shown with double
 13399 brackets, as in regular expression syntax. Several balloters objected to this,
 13400 pointing out that tr does not implement regular expression principles, just bor-
 13401 rows part of the syntax. Consequently, the [:class:] and [=equiv=] should be
 13402 regarded as syntactical elements on a par with [x*n], which is not an RE bracket
 13403 expression.

13404 **4.65 true — Return true value**

13405 **4.65.1 Synopsis**

13406 true

13407 **4.65.2 Description**

13408 The true utility shall return with exit code zero.

13409 **4.65.3 Options**

13410 None.

13411 **4.65.4 Operands**

13412 None.

13413 **4.65.5 External Influences**

13414 **4.65.5.1 Standard Input**

13415 None.

13416 **4.65.5.2 Input Files**

13417 None.

13418 **4.65.5.3 Environment Variables**

13419 None.

13420 **4.65.5.4 Asynchronous Events**

13421 Default.

13422 **4.65.6 External Effects**

13423 **4.65.6.1 Standard Output**

13424 None.

13425 **4.65.6.2 Standard Error**

13426 None.

13427 **4.65.6.3 Output Files**

13428 None.

13429 **4.65.7 Extended Description**

13430 None.

13431 **4.65.8 Exit Status**

13432 The `true` utility always exits with a value of zero.

13433 **4.65.9 Consequences of Errors**

13434 Default.

13435 **4.65.10 Rationale.** *(This subclause is not a part of P1003.2)*

13436 **Examples, Usage**

13437 The `true` utility is typically used in shell scripts. The special built-in utility :
13438 (see 3.14.2) is sometimes more efficient than `true`.

13439 **History of Decisions Made**

13440 The `true` utility has been retained in POSIX.2, even though the shell special
13441 built-in : provides similar functionality, because `true` is widely used in existing
13442 scripts and is less cryptic to novice human script readers.

13443 **4.66 tty — Return user's terminal name**

13444 **4.66.1 Synopsis**

13445 `tty`

13446 *Obsolescent Version:*

13447 `tty -s`

13448 **4.66.2 Description**

13449 The `tty` utility shall write to the standard output the name of the terminal that
 13450 is open as standard input. The name that is used shall be equivalent to the string
 13451 that would be returned by the POSIX.1 {8} *ttyname()* function.

13452 **4.66.3 Options**

13453 The `tty` utility shall conform to the utility argument syntax guidelines described
 13454 in 2.10.2.

13455 The following option shall be supported by the implementation:

13456	<code>-s</code>	(Obsolescent.) Do not write the terminal name. Only the exit
13457		status shall be affected by this option. The terminal status shall
13458		be determined as if the POSIX.1 {8} <i>isatty()</i> function were used.

13459 **4.66.4 Operands**

13460 None.

13461 **4.66.5 External Influences**

13462 **4.66.5.1 Standard Input**

13463 While no input is read from standard input, standard input shall be examined to
 13464 determine whether or not it is a terminal, and/or to determine the name of the
 13465 terminal.

13466 **4.66.5.2 Input Files**

13467 None.

4.66.5.3 Environment Variables

The following environment variables shall affect the execution of `tty`:

LANG	This variable shall determine the locale to use for the locale categories when both LC_ALL and the corresponding environment variable (beginning with LC_) do not specify a locale. See 2.6.
LC_ALL	This variable shall determine the locale to be used to override any values for locale categories specified by the settings of LANG or any environment variables beginning with LC_ .
LC_CTYPE	For the obsolescent version, this variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).
LC_MESSAGES	This variable shall determine the language in which messages should be written.

4.66.5.4 Asynchronous Events

Default.

4.66.6 External Effects

4.66.6.1 Standard Output

If the `-s` option is specified, standard output shall not be used. If the `-s` option is not specified and standard input is a terminal device, a pathname of the terminal as specified by POSIX.1 {8} `ttyname()` shall be written in the following format:

```
"%s\n", <terminal name>
```

Otherwise, a message shall be written indicating that standard input is not connected to a terminal. In the POSIX Locale, the `tty` utility shall use the format:

```
"not a tty\n"
```

4.66.6.2 Standard Error

Used only for diagnostic messages.

4.66.6.3 Output Files

None.

13499 **4.66.7 Extended Description**

13500 None.

13501 **4.66.8 Exit Status**

13502 The `tty` utility shall exit with one of the following values:

- 13503 0 Standard input is a terminal.
13504 1 Standard input is not a terminal.
13505 >1 An error occurred.

13506 **4.66.9 Consequences of Errors**

13507 Default.

13508 **4.66.10 Rationale.** *(This subclause is not a part of P1003.2)*

13509 **Examples, Usage**

13510 This utility checks the status of the file open as standard input against that of a
13511 system-defined set of files. It is possible that no match can be found, or that the
13512 match found need not be the same file as that which was opened for standard
13513 input (although they are the same device).

13514 The `-s` option is useful only if the exit code is wanted. It does not rely on the abil-
13515 ity to form a valid pathname. The `-s` option was made obsolescent because the
13516 same functionality is provided by `test -t 0`, but not dropped completely because
13517 historical scripts depend on this form.

13518 **History of Decisions Made**

13519 The definition of `tty` was made more explicit to explain the difference between a
13520 `tty` and a pathname of a `tty`.

13521 **4.67 umask — Get or set the file mode creation mask**

13522 **4.67.1 Synopsis**

13523 `umask` `[-S]` `[mask]`

13524 **4.67.2 Description**

13525 The `umask` utility shall set the file mode creation mask of the current shell execu-
 13526 tion environment (see 3.12) to the value specified by the *mask* operand. This
 13527 mask shall affect the initial value of the file permission bits of subsequently
 13528 created files.

13529 If the *mask* operand is not specified, the `umask` utility shall write to standard out-
 13530 put the value of the invoking process's file mode creation mask.

13531 **4.67.3 Options**

13532 The `umask` utility shall conform to the utility argument syntax guidelines
 13533 described in 2.10.2.

13534 The following option shall be supported by the implementation:

13535 `-S` Produce symbolic output.

13536 The default output style is unspecified, but shall be recognized on a subsequent
 13537 invocation of `umask` on the same system as a *mask* operand to restore the previ-
 13538 ous file mode creation mask.

13539 **4.67.4 Operands**

13540 The following operand shall be supported by the implementation:

13541 *mask* A string specifying the new file mode creation mask. The string
 13542 is treated in the same way as the *mode* operand described in 4.7.7
 13543 (`chmod` Extended Description).

13544 For a *symbolic_mode* value, the new value of the file mode crea-
 13545 tion mask shall be the logical complement of the file permission
 13546 bits portion of the file mode specified by the *symbolic_mode*
 13547 string.

13548 In a *symbolic_mode* value, the permissions *op* characters `+` and `-`
 13549 shall be interpreted relative to the current file mode creation
 13550 mask; `+` shall cause the bits for the indicated permissions to be
 13551 cleared in the mask; `-` shall cause the bits for the indicated per-
 13552 missions to be set in the mask.

13553 The interpretation of *mode* values that specify file mode bits other
 13554 than the file permission bits is unspecified.

13555 In the obsolescent octal integer form of *mode*, the specified bits
 13556 shall be set in the file mode creation mask.

13557 The file mode creation mask shall be set to the resulting numeric
 13558 value.

13559 As in `chmod`, application use of the octal number form for the
 13560 *mode* values is obsolescent.

13561 The default output of a prior invocation of `umask` on the same sys-
 13562 tem with no operand shall also be recognized as a *mask* operand.
 13563 The use of an operand obtained in this way is not obsolescent,
 13564 even if it is an octal number.

13565 **4.67.5 External Influences**

13566 **4.67.5.1 Standard Input**

13567 None.

13568 **4.67.5.2 Input Files**

13569 None.

13570 **4.67.5.3 Environment Variables**

13571 The following environment variables shall affect the execution of `umask`:

13572	LANG	This variable shall determine the locale to use for the
13573		locale categories when both LC_ALL and the correspond-
13574		ing environment variable (beginning with LC_) do not
13575		specify a locale. See 2.6.
13576	LC_ALL	This variable shall determine the locale to be used to over-
13577		ride any values for locale categories specified by the set-
13578		tings of LANG or any environment variables beginning
13579		with LC_ .
13580	LC_CTYPE	This variable shall determine the locale for the interpreta-
13581		tion of sequences of bytes of text data as characters (e.g.,
13582		single- versus multibyte characters in arguments).
13583	LC_MESSAGES	This variable shall determine the language in which mes-
13584		sages should be written.

4.67.5.4 Asynchronous Events

Default.

4.67.6 External Effects**4.67.6.1 Standard Output**

When the *mask* operand is not specified, the `umask` utility shall write a message to standard output that can later be used as a `umask mask` operand.

If `-S` is specified, the message shall be in the following format:

```
"u=%s,g=%s,o=%s\n", <owner permissions>, <group permissions>,
<other permissions>
```

where the three values shall be combinations of letters from the set {r, w, x}; the presence of a letter shall indicate that the corresponding bit is clear in the file mode creation mask.

If a *mask* operand is specified, there shall be no output written to standard output.

4.67.6.2 Standard Error

Used only for diagnostic messages.

4.67.6.3 Output Files

None.

4.67.7 Extended Description

None.

4.67.8 Exit Status

The `umask` utility shall exit with one of the following values:

- | | |
|----|---|
| 0 | The file mode creation mask was successfully changed, or no <i>mask</i> operand was supplied. |
| >0 | An error occurred. |

13610 **4.67.9 Consequences of Errors**

13611 Default.

13612 **4.67.10 Rationale.** *(This subclause is not a part of P1003.2)*

13613 **Examples, Usage**

13614 Since `umask` affects the current shell execution environment, it is generally pro-
 13615 vided as a shell regular built-in. If it is called in a subshell or separate utility
 13616 execution environment, such as one of the following: 1

```
13617         (umask 002) 1
13618         nohup umask ... 1
13619         find . -exec umask ... \; 1
```

13620 it will not affect the file mode creation mask of the caller's environment. 1

13621 The table mapping octal mode values in 4.7.7 does not require that the symbolic
 13622 constants have those particular values.

13623 In contrast to the negative permission logic provided by the file mode creation
 13624 mask and the octal number form of the *mask* argument, the symbolic form of the
 13625 *mask* argument specifies those permissions that are left alone.

13626 Either of the commands:

```
13627         umask a=rx,ug+w
13628         umask 002
```

13629 sets the mode mask so that subsequently created files have their `S_IWOTH` bit
 13630 cleared.

13631 After setting the mode mask with either of the above commands, the `umask` com-
 13632 mand can be used to write out the current value of the mode mask:

```
13633         $ umask
13634         0002
```

13635 (The output format is unspecified, but historical implementations use the obsoles-
 13636 cent octal integer mode format.)

```
13637         $ umask -S
13638         u=rwx,g=rwx,o=rx
```

13639 Either of these outputs can be used as the mask operand to a subsequent invoca-
 13640 tion of the `umask` utility.

13641 Assuming the mode mask is set as above, the command:

```
13642         umask g-w
```

13643 sets the mode mask so that subsequently created files have their `S_IWGRP`, and
 13644 `S_IWOTH` bits cleared.

13645 The command:

13646 umask -- -w

13647 sets the mode mask so that subsequently created files have all their write bits
13648 cleared. Note that *mask* operands *-r*, *-w*, *-x*, or anything beginning with a
13649 hyphen, must be preceded by *--* to keep it from being interpreted as an option.

13650 History of Decisions Made

13651 The description of the historical utility was modified to allow it to use the sym-
13652 bolic modes of *chmod*. The *-s* option used in earlier drafts was changed to *-S*
13653 because *-s* could be confused with a *symbolic_mode* form of mask referring to the
13654 *S_ISUID* and *S_ISGID* bits.

13655 The default output style is implementation defined to permit implementors to pro-
13656 vide migration to the new symbolic style at the time most appropriate to their
13657 users. Earlier drafts of this standard specified an *-o* flag to force octal mode out-
13658 put. This was dropped because the octal mode may not be sufficient to specify all
13659 of the information that may be present in the file mode creation mask when more
13660 secure file access permission checks are implemented.

13661 It has been suggested that trusted systems developers might appreciate softening
13662 the requirement that the mode mask “affects” the file access permissions, since it
13663 seems access control lists might replace the mode mask to some degree. The
13664 wording has been changed to say that it affects the file permission bits, and leaves
13665 the details of the behavior of how they affect the file access permissions to the
13666 description in POSIX.1 {8}.

13667 **4.68 uname — Return system name**

13668 **4.68.1 Synopsis**

13669 `uname [-amnrsv]`

13670 **4.68.2 Description**

13671 By default, the `uname` utility shall write the operating system name to standard
 13672 output. When options are specified, symbols representing one or more system
 13673 characteristics shall be written to the standard output. The format and contents
 13674 of the symbols are implementation defined. On systems conforming to
 13675 POSIX.1 {8}, the symbols written shall be those supported by the POSIX.1 {8}
 13676 `uname()` function.

13677 **4.68.3 Options**

13678 The `uname` utility shall conform to the utility argument syntax guidelines
 13679 described in 2.10.2.

13680 The following options shall be supported by the implementation:

13681	<code>-a</code>	Behave as though all of the options <code>-mnrsv</code> were specified.
13682	<code>-m</code>	Write the name of the hardware type on which the system is run-
13683		ning to standard output.
13684	<code>-n</code>	Write the name of this node within an implementation-specified
13685		communications network.
13686	<code>-r</code>	Write the current release level of the operating system implemen-
13687		tation.
13688	<code>-s</code>	Write the name of the implementation of the operating system.
13689	<code>-v</code>	Write the current version level of this release of the operating
13690		system implementation.

13691 If no options are specified, the `uname` utility shall write the operating system
 13692 name, as if the `-s` option had been specified.

13693 **4.68.4 Operands**

13694 None.

13695 **4.68.5 External Influences**

13696 **4.68.5.1 Standard Input**

13697 None.

13698 **4.68.5.2 Input Files**

13699 None.

13700 **4.68.5.3 Environment Variables**

13701 The following environment variables shall affect the execution of `uname`:

13702	LANG	This variable shall determine the locale to use for the
13703		locale categories when both LC_ALL and the correspond-
13704		ing environment variable (beginning with LC_) do not
13705		specify a locale. See 2.6.
13706	LC_ALL	This variable shall determine the locale to be used to over-
13707		ride any values for locale categories specified by the set-
13708		tings of LANG or any environment variables beginning
13709		with LC_ .
13710	LC_CTYPE	This variable shall determine the locale for the interpreta-
13711		tion of sequences of bytes of text data as characters (e.g.,
13712		single- versus multibyte characters in arguments).
13713	LC_MESSAGES	This variable shall determine the language in which mes-
13714		sages should be written.

13715 **4.68.5.4 Asynchronous Events**

13716 Default.

13717 **4.68.6 External Effects**

13718 **4.68.6.1 Standard Output**

13719 By default, the output shall be a single line of the following form:

13720 `"%s\n", <sysname>`

13721 If the `-a` option is specified, the output shall be a single line of the following form:

13722 `"%s %s %s %s %s\n", <sysname>, <nodename>, <release>, <version>, <machine>`

13724 Additional implementation-defined symbols may be written; all such symbols
13725 shall be written at the end of the line of output before the `<newline>`.

13726 If options are specified to select different combinations of the symbols, only those
13727 symbols shall be written, in the order shown above for the `-a` option. If a symbol
13728 is not selected for writing, its corresponding trailing `<blank>`s also shall not be
13729 written.

13730 **4.68.6.2 Standard Error**

13731 Used only for diagnostic messages.

13732 **4.68.6.3 Output Files**

13733 None.

13734 **4.68.7 Extended Description**

13735 None.

13736 **4.68.8 Exit Status**

13737 The `uname` utility shall exit with one of the following values:

13738 0 The requested information was successfully written.

13739 >0 An error occurred.

13740 **4.68.9 Consequences of Errors**

13741 Default.

13742 **4.68.10 Rationale.** *(This subclause is not a part of P1003.2)*

13743 **Examples, Usage**

13744 The following command:

13745 `uname -sr`

13746 writes the operating system name and release level, separated by one or more
13747 `<blank>`s.

13748 Note that any of the symbols could include embedded `<space>`s, which may affect
13749 parsing algorithms if multiple options are selected for output.

13750 The node name is typically a name that the system uses to identify itself for inter-
13751 system communication addressing.

13752 History of Decisions Made

13753 It was suggested that this utility cannot be used portably, since the format of the
 13754 symbols is implementation defined. The POSIX.1 {8} working group could not
 13755 achieve consensus on defining these formats in the underlying *uname()* function
 13756 and there is no expectation that POSIX.2 would be any more successful. In any
 13757 event, some applications may still find this historical utility of value. For exam-
 13758 ple, the symbols could be used for system log entries or for comparison with opera-
 13759 tor or user input.

13760 4.69 *uniq* — Report or filter out repeated lines in a file

13761 4.69.1 Synopsis

13762 *uniq* [-c | -d | -u] [-f *fields*] [-s *chars*] [*input_file* [*output_file*]]

13763 *Obsolescent Version:*

13764 *uniq* [-c | -d | -u] [-n] [+m] [*input_file* [*output_file*]]

13765 4.69.2 Description

13766 The *uniq* utility shall read an input file comparing adjacent lines, and write one
 13767 copy of each input line on the output. The second and succeeding copies of
 13768 repeated adjacent input lines shall not be written.

13769 Repeated lines in the input shall not be detected if they are not adjacent.

13770 4.69.3 Options

13771 The *uniq* utility shall conform to the utility argument syntax guidelines
 13772 described in 2.10.2; the obsolescent version does not, as one of the options begins
 13773 with + and the *-m* and *+n* options do not have option letters.

13774 The following options shall be supported by the implementation:

13775 -c Precede each output line with a count of the number of times the
 13776 line occurred in the input.

13777 -d Suppress the writing of lines that are not repeated in the input.

13778 -f *fields* Ignore the first *fields* fields on each input line when doing com-
 13779 parisons, where *fields* shall be a positive decimal integer. A field
 13780 is the maximal string matched by the basic regular expresssion:

13781 [[[:blank:]]]*[^[[:blank:]]]*

13782 If the *fields* option-argument specifies more fields than appear on
 13783 an input line, a null string shall be used for comparison.

- 13784 `-s chars` Ignore the first *chars* characters when doing comparisons, where
 13785 *chars* shall be a positive decimal integer. If specified in conjunc-
 13786 tion with the `-f` option, the first *chars* characters after the first
 13787 *fields* fields shall be ignored. If the *chars* option-argument
 13788 specifies more characters than remain on an input line, a null
 13789 string shall be used for comparison.
- 13790 `-u` Suppress the writing of lines that are repeated in the input.
- 13791 `-n` (Obsolescent.) Equivalent to `-f fields` with *fields* set to *n*.
- 13792 `+m` (Obsolescent.) Equivalent to `-s chars` with *chars* set to *m*.

13793 **4.69.4 Operands**

13794 The following operands shall be supported by the implementation:

- 13795 *input_file* A pathname of the input file. If the *input_file* operand is not
 13796 specified, or if the *input_file* is `-`, the standard input shall be
 13797 used.
- 13798 *output_file* A pathname of the output file. If the *output_file* operand is not
 13799 specified, the standard output shall be used. The results are
 13800 unspecified if the file named by *output_file* is the file named by
 13801 *input_file*.

13802 **4.69.5 External Influences**

13803 **4.69.5.1 Standard Input**

13804 The standard input shall be used only if no *input_file* operand is specified or if
 13805 *input_file* is `-`. See Input Files.

13806 **4.69.5.2 Input Files**

13807 The input file shall be a text file.

13808 **4.69.5.3 Environment Variables**

13809 The following environment variables shall affect the execution of `uniq`:

- 13810 **LANG** This variable shall determine the locale to use for the
 13811 locale categories when both **LC_ALL** and the correspond-
 13812 ing environment variable (beginning with **LC_**) do not
 13813 specify a locale. See 2.6.
- 13814 **LC_ALL** This variable shall determine the locale to be used to over-
 13815 ride any values for locale categories specified by the set-
 13816 tings of **LANG** or any environment variables beginning
 13817 with **LC_**.

13818 **LC_CTYPE** This variable shall determine the locale for the interpreta-
 13819 tion of sequences of bytes of text data as characters (e.g.,
 13820 single- versus multibyte characters in arguments and
 13821 input files) and which characters constitute a <blank> in
 13822 the current locale.

13823 **LC_MESSAGES** This variable shall determine the language in which mes-
 13824 sages should be written.

13825 **4.69.5.4 Asynchronous Events**

13826 Default.

13827 **4.69.6 External Effects**

13828 **4.69.6.1 Standard Output**

13829 The standard output shall be used only if no *output_file* operand is specified. See
 13830 Output Files.

13831 **4.69.6.2 Standard Error**

13832 Used only for diagnostic messages.

13833 **4.69.6.3 Output Files**

13834 If the *-c* option is specified, the output file shall be empty or each line will be of
 13835 the form:

13836 "%d %s", <number of duplicates>, <line>

13837 otherwise, the output file will be empty or each line will be of the form:

13838 "%s", <line>

13839 **4.69.7 Extended Description**

13840 None.

13841 **4.69.8 Exit Status**

13842 The *uniq* utility shall exit with one of the following values:

13843 0 The utility executed successfully.

13844 >0 An error occurred.

13845 4.69.9 Consequences of Errors

13846 Default.

13847 4.69.10 Rationale. *(This subclause is not a part of P1003.2)*

13848 Examples, Usage

13849 Some historical implementations have limited lines to be 1080 bytes in length,
13850 which will not meet the implied {LINE_MAX} limit.

13851 The `sort` utility (see 4.58) can be used to cause repeated lines to be adjacent in
13852 the input file.

13853 The following input file data (but flushed left) was used for a test series on `uniq`:

```
13854     #01 foo0 bar0 fool bar1
13855     #02 bar0 fool bar1 fool
13856     #03 foo0 bar0 fool bar1
13857     #04
13858     #05 foo0 bar0 fool bar1
13859     #06 foo0 bar0 fool bar1
13860     #07 bar0 fool bar1 foo0
```

13861 What follows is a series of test invocations of the `uniq` utility that use a mixture
13862 of `uniq`'s options against the input file data. These tests verify the meaning of
13863 *adjacent*. The `uniq` utility views the input data as a sequence of strings delim-
13864 ited by `\n`. Accordingly, for the *fields**th* member of the sequence, `uniq` interprets
13865 unique or repeated adjacent lines strictly relative to the *fields*+1th member.

13866 This first example tests the line counting option, comparing each line of the input
13867 file data starting from the second field:

```
13868     uniq -c -f 1 uniq_0I.t
13869     1 #01 foo0 bar0 fool bar1
13870     1 #02 bar0 fool bar1 foo0
13871     1 #03 foo0 bar0 fool bar1
13872     1 #04
13873     2 #05 foo0 bar0 fool bar1
13874     1 #07 bar0 fool bar1 foo0
```

13875 The number 2, prefixing the fifth line of output, signifies that the `uniq` utility
13876 detected a pair of repeated lines. Given the input data, this can only be true
13877 when `uniq` is run using the `-f 1` option (which causes `uniq` to ignore the first
13878 field on each input line).

13879 The second example tests the option to suppress unique lines, comparing each
13880 line of the input file data starting from the second field:

```
13881     uniq -d -f 1 uniq_0I.t
13882     #05 foo0 bar0 fool bar1
```

13883 This test suppresses repeated lines, comparing each line of the input file data
13884 starting from the second field:

```

13885      uniq -u -f 1 uniq_0I.t
13886      #01 foo0 bar0 fool bar1
13887      #02 bar0 fool bar1 fool
13888      #03 foo0 bar0 fool bar1
13889      #04
13890      #07 bar0 fool bar1 foo0

```

13891 This suppresses unique lines, comparing each line of the input file data starting
 13892 from the third character:

```

13893      uniq -d -s 2 uniq_0I.t

```

13894 In the last example, the `uniq` utility found no input matching the above criteria.

13895 History of Decisions Made

13896 The `-f` and `-s` options were added to replace the obsolescent `-n` and `+m` options
 13897 so that `uniq` could meet the syntax guidelines in an upward-compatible way.

13898 The output specifications in Output Files do not show a terminating `<newline>`
 13899 because they both specify `<line>`, which includes its own `<newline>` (because of
 13900 the definition of *line*).

13901 4.70 wait — Await process completion

13902 4.70.1 Synopsis

```

13903 wait [pid ...]

```

13904 4.70.2 Description

13905 When an asynchronous list (see 3.9.3.1) is started by the shell, the process ID of
 13906 the last command in each element of the asynchronous list shall become known in 1
 13907 the current shell execution environment; see 3.12.

13908 If the `wait` utility is invoked with no operands, it shall wait until all process IDs
 13909 known to the invoking shell have terminated and exit with a zero exit status.

13910 If one or more *pid* operands are specified that represent known process IDs, the
 13911 `wait` utility shall wait until all of them have terminated. If one or more *pid*
 13912 operands are specified that represent unknown process IDs, `wait` shall treat them
 13913 as if they were known process IDs that exited with exit status 127. The exit
 13914 status returned by the `wait` utility shall be the exit status of the process
 13915 requested by the last *pid* operand.

13916 The known process IDs are applicable only for invocations of `wait` in the current
 13917 shell execution environment.

13918 **4.70.3 Options**

13919 None.

13920 **4.70.4 Operands**

13921 The following operand shall be supported by the implementation:

13922	<i>pid</i>	The unsigned decimal integer process ID of a command, for which
13923		the utility is to wait for the termination.

13924 **4.70.5 External Influences**

13925 **4.70.5.1 Standard Input**

13926 None.

13927 **4.70.5.2 Input Files**

13928 None.

13929 **4.70.5.3 Environment Variables**

13930 The following environment variables shall affect the execution of `wait`:

13931	LANG	This variable shall determine the locale to use for the
13932		locale categories when both LC_ALL and the correspond-
13933		ing environment variable (beginning with LC_) do not
13934		specify a locale. See 2.6.
13935	LC_ALL	This variable shall determine the locale to be used to over-
13936		ride any values for locale categories specified by the set-
13937		tings of LANG or any environment variables beginning
13938		with LC_ .
13939	LC_CTYPE	This variable shall determine the locale for the interpreta-
13940		tion of sequences of bytes of text data as characters (e.g.,
13941		single- versus multibyte characters in arguments).
13942	LC_MESSAGES	This variable shall determine the language in which mes-
13943		sages should be written.

13944 **4.70.5.4 Asynchronous Events**

13945 Default.

13946 **4.70.6 External Effects**

13947 **4.70.6.1 Standard Output**

13948 None.

13949 **4.70.6.2 Standard Error**

13950 Used only for diagnostic messages.

13951 **4.70.6.3 Output Files**

13952 None.

13953 **4.70.7 Extended Description**

13954 None.

13955 **4.70.8 Exit Status**

13956 If one or more operands were specified, all of them have terminated or were not
 13957 known by the invoking shell, and the status of the last operand specified is
 13958 known, then the exit status of `wait` shall be the exit status information of the
 13959 command indicated by the last operand specified. If the process terminated
 13960 abnormally due to the receipt of a signal, the exit status shall be greater than 128
 13961 and shall be distinct from the exit status generated by other signals, but the exact
 13962 value is unspecified. (See the `kill -l` option in 4.32.) Otherwise, the `wait` util-
 13963 ity shall exit with one of the following values:

13964	0	The <code>wait</code> utility was invoked with no operands and all process IDs
13965		known by the invoking shell have terminated.
13966	1–126	The <code>wait</code> utility detected an error.
13967	127	The command identified by the last <i>pid</i> operand specified is
13968		unknown.

13969 **4.70.9 Consequences of Errors**

13970 Default.

13971 **4.70.10 Rationale.** *(This subclause is not a part of P1003.2)*

13972 Examples, Usage

13973 On most implementations, `wait` is a shell built-in. If it is called in a subshell or 1
13974 separate utility execution environment, such as one of the following: 1

```
13975     (wait) 1
13976     nohup wait ... 1
13977     find . -exec wait ... \; 1
```

13978 it will return immediately because there will be no known process IDs to wait for 1
13979 in those environments. 1

13980 Although the exact value used when a process is terminated by a signal is
13981 unspecified, if it is known that a signal terminated a process, a script can still
13982 reliably figure out which signal using `kill` as shown by the following script:

```
13983     sleep 1000&
13984     pid=$!
13985     kill -kill $pid
13986     wait $pid
13987     echo $pid was terminated by a SIG$(kill -l $?) signal.
```

13988 Historical implementations of interactive shells have discarded the exit status of
13989 terminated background processes before each shell prompt. Therefore, the status
13990 of background processes was usually lost unless it terminated while `wait` was
13991 waiting for it. This could be a serious problem when a job that was expected to
13992 run for a long time actually terminated quickly with a syntax or initialization
13993 error because the exit status returned was usually zero if the requested process ID
13994 was not found. POSIX.2 requires the implementation to keep the status of ter-
13995 minated jobs available until the status is requested, so that scripts like:

```
13996     j1&
13997     p1=$!
13998     j2&
13999     wait $p1
14000     echo Job 1 exited with status $?
14001     wait $!
14002     echo Job 2 exited with status $?
```

14003 will work without losing status on any of the jobs. The shell is allowed to discard
14004 the status of any process that it determines the application cannot get the process
14005 ID from the shell. It is also required to remember only {CHILD_MAX} number of 1
14006 processes in this way. Since the only way to get the process ID from the shell is by
14007 using the `!` shell parameter, the shell is allowed to discard the status of an asyn-
14008 chronous list if `$!` was not referenced before another asynchronous list was
14009 started. (This means that the shell only has to keep the status of the last asyn-
14010 chronous list started if the application did not reference `$!`. If the implementa-
14011 tion of the shell is smart enough to determine that a reference to `$!` was not
14012 “saved” anywhere that the application can retrieve it later, it can use this infor-
14013 mation to trim the list of saved information. Note also that a successful call to
14014 `wait` with no operands discards the exit status of all asynchronous lists.)

This new functionality was added because it is needed to accurately determine the exit status of any asynchronous list. The only compatibility problem that this change creates is for a script like:

```
while sleep 60
do
    job&
    echo Job started $(date) as $!
done
```

which will cause the shell to keep track of all of the jobs started until the script terminates or runs out of memory. This would not be a problem if the loop did not reference `$!` or if the script would occasionally `wait` for jobs it started.

If the exit status of `wait` is greater than 128, there is no way for the application to know if the waited for process exited with that value or was killed by a signal. Since most utilities exit with small values, there is seldom any ambiguity. Even in the ambiguous cases, most applications just need to know that the asynchronous job failed; it does not matter whether it detected an error and failed or was killed and did not complete its job normally.

History of Decisions Made

The description of `wait` does not refer to the `waitpid()` function from POSIX.1 {8}, because that would needlessly overspecify this interface. However, the wording requires that `wait` is required to wait for an explicit process when it is given an argument, so that the status information of other processes is not consumed. Historical implementations use POSIX.1 {8} `wait()` until `wait()` returns the requested process ID or finds that the requested process does not exist. Because this means that a shell script could not reliably get the status of all background children if a second background job was ever started before the first job finished, it is recommended that the `wait` utility use a method such as the functionality provided by the `waitpid()` function in POSIX.1 {8}.

The ability to wait for multiple *pid* operands was adopted from the KornShell at the request of ballot comments and objections.

Some implementations of `wait` support waiting for asynchronous lists identified by the use of job identifiers. For example, `wait %1` would wait for the first background job. This standard does not address job control issues, but allows these features to be added as extensions. Job control facilities will be provided by the UPE.

14050 **4.71 wc — Word, line, and byte count**

14051 **4.71.1 Synopsis**

14052 `wc [-clw] [file ...]`

14053 **4.71.2 Description**

14054 The `wc` utility shall read one or more input files and, by default, write the number
 14055 of <newline>s, words, and bytes contained in each input file to the standard out-
 14056 put.

14057 The utility also shall write a total count for all named files, if more than one input
 14058 file is specified.

14059 The `wc` utility shall consider a *word* to be a nonzero-length string of characters
 14060 delimited by white space.

14061 **4.71.3 Options**

14062 The `wc` utility shall conform to the utility argument syntax guidelines described
 14063 in 2.10.2.

14064 The following options shall be supported by the implementation:

14065 `-c` Write to the standard output the number of bytes in each input
 14066 file.

14067 `-l` Write to the standard output the number of <newline>s in each
 14068 input file.

14069 `-w` Write to the standard output the number of words in each input
 14070 file.

14071 When any option is specified, `wc` shall report only the information requested by
 14072 the specified option(s).

14073 **4.71.4 Operands**

14074 The following operand shall be supported by the implementation:

14075 *file* A pathname of an input file. If no *file* operands are specified, the
 14076 standard input shall be used.

14077 **4.71.5 External Influences**

14078 **4.71.5.1 Standard Input**

14079 The standard input shall be used only if no *file* operands are specified. See Input
14080 Files.

14081 **4.71.5.2 Input Files**

14082 The input files may be of any type.

14083 **4.71.5.3 Environment Variables**

14084 The following environment variables shall affect the execution of `wc`:

14085	LANG	This variable shall determine the locale to use for the
14086		locale categories when both LC_ALL and the correspond-
14087		ing environment variable (beginning with LC_) do not
14088		specify a locale. See 2.6.

14089	LC_ALL	This variable shall determine the locale to be used to over-
14090		ride any values for locale categories specified by the set-
14091		tings of LANG or any environment variables beginning
14092		with LC_ .

14093	LC_CTYPE	This variable shall determine the locale for the interpreta-
14094		tion of sequences of bytes of text data as characters (e.g.,
14095		single- versus multibyte characters in arguments and
14096		input files) and which characters are defined as “white
14097		space” characters.

14098	LC_MESSAGES	This variable shall determine the language in which mes-
14099		sages should be written.

14100 **4.71.5.4 Asynchronous Events**

14101 Default.

14102 **4.71.6 External Effects**

14103 **4.71.6.1 Standard Output**

14104 By default, the standard output shall contain a line for each input file of the form:

14105 "%d %d %d %s\n", <newlines>, <words>, <bytes>, <file>

14106 If any options are specified and the `-l` option is not specified, the number of
14107 <newline>s shall not be written.

14108 If any options are specified and the `-w` option is not specified, the number of words
14109 shall not be written.

14110 If any options are specified and the `-c` option is not specified, the number of bytes
14111 shall not be written.

14112 If no input *file* operands are specified, no name shall be written and no `<blank>`s
14113 preceding the pathname shall be written.

14114 If more than one input *file* operand is specified, an additional line shall be writ-
14115 ten, of the same format as the other lines, except that the word `total` (in the
14116 POSIX Locale) shall be written instead of a pathname and the total of each column
14117 shall be written as appropriate. Such an additional line, if any, shall be written
14118 at the end of the output.

14119 **4.71.6.2 Standard Error**

14120 Used only for diagnostic messages.

14121 **4.71.6.3 Output Files**

14122 None.

14123 **4.71.7 Extended Description**

14124 None.

14125 **4.71.8 Exit Status**

14126 The `wc` utility shall exit with one of the following values:

14127 0 Successful completion.

14128 >0 An error occurred.

14129 **4.71.9 Consequences of Errors**

14130 Default.

14131 **4.71.10 Rationale.** *(This subclause is not a part of P1003.2)*

14132 **Examples, Usage**

14133 None.

14134 History of Decisions Made

14135 The output file format pseudo-*printf()* string was derived from the HP-UX version
14136 of *wc*; the System V version:

14137 "%7d%7d%7d %s\n"

14138 produces possibly ambiguous and unparsable results for very large files, as it
14139 assumes no number will exceed six digits.

14140 Some historical implementations use only <space>, <tab>, and <newline> as
14141 word separators. The equivalent of the C Standard {7} *isspace()* function is more
14142 appropriate.

14143 The *-c* option stands for “character” count, even though it counts bytes. This
14144 stems from the sometimes erroneous historical view that bytes and characters are
14145 the same size.

14146 Earlier drafts only specified the results when input files were text files. The
14147 current specification more closely matches existing practice. (Bytes, words, and
14148 <newline>s are counted separately and the results are written when an end-of-
14149 file is detected.)

14150 Historical implementations of the *wc* utility only accepted one argument to specify
14151 the options *-c*, *-l*, and *-w*. Some of them also had multiple occurrences of an
14152 option cause the corresponding count to be output multiple times and having the
14153 order of specification of the options affect the order of the fields on output, but did
14154 not document either of these. Because common usage either specifies no options
14155 or only one option and because none of this was documented, the changes
14156 required by this standard should not break many existing applications (and does
14157 not break any historical portable applications.)

14158 4.72 **xargs** — Construct argument list(s) and invoke utility

14159 4.72.1 Synopsis

14160 `xargs [-t] [-n number [-x]] [-s size] [utility [argument ...]]`

14161 4.72.2 Description

14162 The `xargs` utility shall construct a command line consisting of the *utility* and
 14163 *argument* operands specified followed by as many arguments read in sequence
 14164 from standard input as will fit in length and number constraints specified by the
 14165 options. The `xargs` utility shall then invoke the constructed command line and
 14166 wait for its completion. This sequence shall be repeated until an end-of-file condi-
 14167 tion is detected on standard input or an invocation of a constructed command line
 14168 returns an exit status of 255.

1

14169 Arguments in the standard input shall be separated by unquoted <blank>s, or
 14170 unescaped <blank>s or <newline>s. A string of zero or more nondouble-quote
 14171 (") and non-<newline> characters can be quoted by enclosing them in double-
 14172 quotes. A string of zero or more nonapostrophe (') and non-<newline> charac-
 14173 ters can be quoted by enclosing them in apostrophes. Any unquoted character can
 14174 be escaped by preceding it with a backslash. The *utility* shall be executed one or
 14175 more times until the end-of-file is reached. The results are unspecified if the util-
 14176 ity named by *utility* attempts to read from its standard input.

14177 The generated command line length shall be the sum of the size in bytes of the
 14178 utility name and each argument treated as strings, including a null byte termina-
 14179 tor for each of these strings. The `xargs` utility shall limit the command line
 14180 length such that when the command line is invoked, the combined argument and
 14181 environment lists (see the *exec* family of functions in POSIX.1 {8} 3.1.2) shall not
 14182 exceed {ARG_MAX}–2048 bytes. Within this constraint, if neither the `-n` nor the
 14183 `-s` option is specified, the default command line length shall be at least
 14184 {LINE_MAX}.

14185 4.72.3 Options

14186 The `xargs` utility shall conform to the utility argument syntax guidelines
 14187 described in 2.10.2.

14188 The following options shall be supported by the implementation:

- 14189 `-n number` Invoke *utility* using as many standard input arguments as possi-
 14190 ble, up to *number* (a positive decimal integer) arguments max-
 14191 imum. Fewer arguments shall be used if:
 - 14192 — The command line length accumulated exceeds the size
 14193 specified by the `-s` option (or {LINE_MAX} if there is no `-s`
 14194 option), or

- 14195 — The last iteration has fewer than *number*, but not zero,
 14196 operands remaining.
- 14197 *-s size* Invoke *utility* using as many standard input arguments as possi-
 14198 ble yielding a command line length less than *size* (a positive
 14199 decimal integer) bytes. Fewer arguments shall be used if:
- 14200 — The total number of arguments exceeds that specified by the
 14201 *-n* option, or
- 14202 — End of file is encountered on standard input before *size* bytes
 14203 are accumulated.
- 14204 Implementations shall support values of *size* up to at least
 14205 {LINE_MAX} bytes, provided that the constraints specified in
 14206 4.72.2 are met. It shall not be considered an error if a value
 14207 larger than that supported by the implementation or exceeding
 14208 the constraints specified in 4.72.2 is given; *xargs* shall use the
 14209 largest value it supports within the constraints.
- 14210 *-t* Enable trace mode. Each generated command line shall be writ-
 14211 ten to standard error just prior to invocation.
- 14212 *-x* Terminate if a command line containing *number* arguments (see
 14213 the *-n* option above) will not fit in the implied or specified size
 14214 (see the *-s* option above).

14215 4.72.4 Operands

14216 The following operands shall be supported by the implementation:

- 14217 *utility* The name of the utility to be invoked, found by search path using
 14218 the **PATH** environment variable, described in 2.6. If *utility* is
 14219 omitted, the default shall be the *echo* utility (see 4.19). If the
 14220 *utility* operand names any of the special built-in utilities in 3.14,
 14221 the results are undefined.
- 14222 *argument* An initial option or operand for the invocation of *utility*.

14223 4.72.5 External Influences

14224 4.72.5.1 Standard Input

14225 The standard input shall be a text file. The results are unspecified if an end-of-
 14226 file condition is detected immediately following an escaped <newline>.

14227 **4.72.5.2 Input Files**

14228 None.

14229 **4.72.5.3 Environment Variables**

14230 The following environment variables shall affect the execution of `xargs`:

14231	LANG	This variable shall determine the locale to use for the
14232		locale categories when both LC_ALL and the correspond-
14233		ing environment variable (beginning with LC_) do not
14234		specify a locale. See 2.6.
14235	LC_ALL	This variable shall determine the locale to be used to over-
14236		ride any values for locale categories specified by the set-
14237		tings of LANG or any environment variables beginning
14238		with LC_ .
14239	LC_CTYPE	This variable shall determine the locale for the interpreta-
14240		tion of sequences of bytes of text data as characters (e.g.,
14241		single- versus multibyte characters in arguments and
14242		input files).
14243	LC_MESSAGES	This variable shall determine the language in which mes-
14244		sages should be written.

14245 **4.72.5.4 Asynchronous Events**

14246 Default.

14247 **4.72.6 External Effects**

14248 Any external effects are a result of the invocation of the utility *utility*, in a
14249 manner specified by that utility.

14250 **4.72.6.1 Standard Output**

14251 None.

14252 **4.72.6.2 Standard Error**

14253 Used for diagnostic messages and the `-t` option. If the `-t` option is specified, the
14254 *utility* and its constructed argument list shall be written to standard error, as it
14255 will be invoked, prior to invocation.

14256 **4.72.6.3 Output Files**

14257 None.

14258 **4.72.7 Extended Description**

14259 None.

14260 **4.72.8 Exit Status**

14261 The `xargs` utility shall exit with one of the following values:

14262	0	All invocations of <i>utility</i> returned exit status zero.	
14263	1–125	A command line meeting the specified requirements could not be	1
14264		assembled, one or more of the invocations of <i>utility</i> returned a	1
14265		nonzero exit status, or some other error occurred.	1
14266	126	The utility specified by <i>utility</i> was found but could not be invoked.	1
14267	127	The utility specified by <i>utility</i> could not be found.	1

14268 **4.72.9 Consequences of Errors**

14269 If a command line meeting the specified requirements cannot be assembled, the
 14270 utility cannot be invoked, an invocation of the utility is terminated by a signal, or
 14271 an invocation of the utility exits with exit status 255, the `xargs` utility shall write
 14272 a diagnostic message and exit without processing any remaining input.

14273 **4.72.10 Rationale.** *(This subclause is not a part of P1003.2)*

14274 **Examples, Usage**

14275 The `xargs` utility is usually found only in System V-based systems; BSD systems
 14276 provide an `apply` utility that provides functionality similar to `xargs -n number`.
 14277 The *SVID* lists `xargs` as a software development extension; POSIX.2 does not
 14278 share the view that it is used only for development, and therefore it is not
 14279 optional.

14280 Note that input is parsed as lines and <blank>s separate arguments. If `xargs` is
 14281 used to bundle output of commands like `find dir -print` or `ls` into commands
 14282 to be executed, unexpected results are likely if any file names contain any
 14283 <blank>s or <newline>s. This can be fixed by using `find` to call a script that
 14284 converts each file found into a quoted string that is then piped to `xargs`. Note
 14285 that the quoting rules used by `xargs` are not the same as in the shell. They were
 14286 not made consistent here because existing applications depend on the current
 14287 rules and the shell syntax is not fully compatible with it. An easy rule that can be
 14288 used to transform any string into a quoted form that `xargs` will interpret
 14289 correctly is to precede each character in the string with a backslash.

14290 The following command will combine the output of the parenthesized commands
14291 onto one line, which is then written to the end of file `log`:

```
14292      (logname; date; printf "%s\n" "$0 $*") | xargs >>log
```

14293 The following command will invoke `diff` with successive pairs of arguments ori-
14294 ginally typed as command line arguments (assuming there are no embedded
14295 <blank>s in the elements of the original argument list):

```
14296      printf "%s\n" "$*" | xargs -n 2 -x diff
```

14297 On implementations with a large value for `{ARG_MAX}`, `xargs` may produce com-
14298 mand lines longer than `{LINE_MAX}`. For invocation of utilities, this is not a prob-
14299 lem. If `xargs` is being used to create a text file, users should explicitly set the
14300 maximum command line length with the `-s` option.

14301 History of Decisions Made

14302 The list of options has been scaled down extensively. As it had stood, the `xargs`
14303 utility did not exhibit an economy of powerful, modular, or extensible functional-
14304 ity.

14305 The classic application of the `xargs` utility is in conjunction with the `find` utility
14306 to reduce the number of processes launched by a simplistic use of the `find`
14307 `-exec` combination. The `xargs` utility is also used to enforce an upper limit on
14308 memory required to launch a process. With this basis in mind, POSIX.2 selected
14309 only the minimal features required.

14310 The `-n number` option was classically used to evoke a utility using pairs of
14311 operands, yet the general case has problems when *utility* spawns child processes
14312 of its own. The `xargs` utility can sap resources from these children, especially
14313 those sharing the parent's environment.

14314 The command, `env`, `nohup`, and `xargs` utilities have been specified to use exit
14315 code 127 if an error occurs so that applications can distinguish “failure to find a
14316 utility” from “invoked utility exited with an error indication.” The value 127 was
14317 chosen because it is not commonly used for other meanings; most utilities use
14318 small values for “normal error conditions” and the values above 128 can be con-
14319 fused with termination due to receipt of a signal. The value 126 was chosen in a
14320 similar manner to indicate that the utility could be found, but not invoked. Some
14321 scripts produce meaningful error messages differentiating the 126 and 127 cases.
14322 The distinction between exit codes 126 and 127 is based on KornShell practice
14323 that uses 127 when all attempts to *exec* the utility fail with `[ENOENT]`, and uses
14324 126 when any attempt to *exec* the utility fails for any other reason.

14325 Although the 255 exit status is mostly an accident of historical implementations,
14326 it allows a utility being used by `xargs` to tell `xargs` to terminate if it knows no
14327 further invocations using the current data stream will succeed. Any nonzero exit
14328 status from a utility will fall into the 1–125 range when `xargs` exits. There is no
14329 statement of how the various nonzero utility exit status codes are accumulated by
14330 `xargs`. The value could be the addition of all codes, their highest value, the last
14331 one received, or a single value such as 1. Since no algorithm is arguably better
14332 than the others, and since many of the POSIX.2 standard utilities say little more

14333 (portably) than “pass/fail,” no new algorithm was invented.

14334 Several other `xargs` options were withdrawn because simple alternatives already
14335 exist within the standard. For example, the `-eeofstr` option has a `sed` work
14336 around. The `-ireplstr` option can be just as efficiently performed using a shell
14337 for loop. Since `xargs` will `exec()` with each input line, the `-i` option will usually
14338 not exploit `xargs`’s grouping capabilities.

14339 The `-s` option was reinstated since many of the balloters on Draft 8 felt that it
14340 was preferable to the `-r` option invented for that draft that required the imple-
14341 mentation to use `{ARG_MAX}` – *size* bytes for command lines.

14342 The requirement that `xargs` never produce command lines such that invocation
14343 of *utility* is within 2048 bytes of hitting the POSIX.1 {8} `exec {ARG_MAX}` limita-
14344 tions is intended to guarantee that the invoked utility has a little bit of room to
14345 modify its environment variables and command line arguments and still be able
14346 to invoke another utility. Note that the minimum `{ARG_MAX}` allowed by
14347 POSIX.1 {8} is 4096 and the minimum value allowed by POSIX.2 is 2048; therefore,
14348 the 2048-byte difference seems reasonable. Note, however, that `xargs` may never
14349 be able to invoke a utility if the environment passed in to `xargs` comes close to
14350 using `{ARG_MAX}` bytes.

14351 The version of `xargs` required by POSIX.2 is required to wait for the completion of
14352 the invoked command before invoking another command. This was done because
14353 existing scripts using `xargs` assumed sequential execution. Implementations
14354 wanting to provide parallel operation of the invoked utilities are encouraged to
14355 add an option enabling parallel invocation, but should still wait for termination of
14356 all of the children before `xargs` terminates normally.

Section 5: User Portability Utilities Option

1 *Editor's Note: This empty section is placeholder for a future revision (the User Por-*
2 *tability Extension, P1003.2a) to contain descriptions of utilities that are suitable*
3 *for user portability on asynchronous character terminals. P1003.2a is currently*
4 *balloting within the IEEE. Contact the IEEE Standards Office to obtain a copy of*
5 *the latest draft.*

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

Section 6: Software Development Utilities Option

This section describes utilities used for the development of applications, including compilation or translation of source code, the creation and maintenance of library archives, and the maintenance of groups of interdependent programs.

The utilities described in this section may be provided by the conforming system; however, any system claiming conformance to the **Software Development Utilities Option** shall provide all of the utilities described here.

6.1 ar — Create and maintain library archives

6.1.1 Synopsis

```
ar -d [-v] archive file ...
ar -p [-v] archive [file ...]
ar -r [-cuv] archive file ...
ar -t [-v] archive [file ...]
ar -x [-v] archive [file ...]
```

6.1.2 Description

The `ar` utility can be used to create and maintain groups of files combined into an archive. Once an archive has been created, new files can be added, and existing files can be extracted, deleted, or replaced. When an archive consists entirely of valid object files, the implementation shall format the archive so that it is usable as a library for link editing (see A.1 and C.2). When some of the archived files are not valid object files, the suitability of the archive for library use is undefined.

All *file* operands can be pathnames. However, files within archives shall be named by a filename, which is the last component of the pathname used when the file was entered into the archive. The comparison of *file* operands to the names of files in archives shall be performed by comparing the last component of the operand to the name of the archive file.

It is unspecified whether multiple files in the archive may be identically named. In the case of such files, however, each *file* operand shall match only the first

archive file having a name that is the same as the last component of the *file* operand.

6.1.3 Options

The `ar` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

- `-c` Suppress the diagnostic message that is written to standard error by default when the archive file *archive* is created.
- `-d` Delete *file(s)* from *archive*.
- `-p` Write the contents of the *file(s)* from *archive* to the standard output. If no *file(s)* are specified, the contents of all files in the archive shall be written in the order of the archive.
- `-r` Replace or add *file(s)* to *archive*. If the archive named by *archive* does not exist, a new archive file shall be created and a diagnostic message shall be written to standard error (unless the `-c` option is specified). If no *file(s)* are specified and the *archive* exists, the results are undefined. Files that replace existing files shall not change the order of the archive. Files that do not replace existing files shall be appended to the archive.
- `-t` Write a table of contents of *archive* to the standard output. The files specified by the *file* operands shall be included in the written list. If no *file* operands are specified, all files in *archive* shall be included in the order of the archive.
- `-u` Update older files. When used with the `-r` option, files within the archive will be replaced only if the corresponding *file* has a modification time that is at least as new as the modification time of the file within the archive.
- `-v` Give verbose output. When used with the option characters `-d`, `-r`, or `-x`, write a detailed file-by-file description of the archive creation and maintenance activity, as described in 6.1.6.1.
- When used with `-p`, write the name of the file to the standard output before writing the file itself to the standard output, as described in 6.1.6.1.
- When used with `-t`, include a long listing of information about the files within the archive, as described in 6.1.6.1.
- `-x` Extract the files named by the *file* operands from *archive*. The contents of the archive file shall not be changed. If no *file* operands are given, all files in the archive shall be extracted. If the filename of a file extracted from the archive is longer than that supported in the directory to which it is being extracted, the

68 results are undefined. The modification time of each file
 69 extracted shall be set to the time the file is extracted from the
 70 archive.

71 6.1.4 Operands

72 The following operands shall be supported by the implementation:

73	<i>archive</i>	A pathname of the archive file.
74	<i>file</i>	A pathname. Only the last component shall be used when comparing against the names of files in the archive. If two or more
75		<i>file</i> operands have the same last pathname component
76		(basename), the results are unspecified. The implementation's
77		archive format shall not truncate valid filenames of files added to,
78		or replaced in, the archive.
79		

80 6.1.5 External Influences

81 6.1.5.1 Standard Input

82 None.

83 6.1.5.2 Input Files

84 The input file named by *archive* shall be a file in the format created by `ar -r`.

85 6.1.5.3 Environment Variables

86 The following environment variables shall affect the execution of `ar`:

87	LANG	This variable shall determine the locale to use for the
88		locale categories when both LC_ALL and the correspond-
89		ing environment variable (beginning with LC_) do not
90		specify a locale. See 2.6.
91	LC_ALL	This variable shall determine the locale to be used to over-
92		ride any values for locale categories specified by the set-
93		tings of LANG or any environment variables beginning
94		with LC_ .
95	LC_CTYPE	This variable shall determine the locale for the interpreta-
96		tion of sequences of bytes of text data as characters (e.g.,
97		single- versus multibyte characters in arguments).
98	LC_MESSAGES	This variable shall determine the language in which mes-
99		sages should be written.

100 **LC_TIME** This variable shall determine the format and content for
101 date and time strings written by `ar`.

102 **6.1.5.4 Asynchronous Events**

103 Default.

104 **6.1.6 External Effects**

105 **6.1.6.1 Standard Output**

106 If the `-d` option is used with the `-v` option, the standard output format is:

107 `"d - %s\n", <file>`

108 where *file* is the operand specified on the command line.

109 If the `-p` option is used with the `-v` option, `ar` shall precede the contents of each
110 file with:

111 `"\n<%s>\n\n", <file>`

112 where *file* is the operand specified on the command line, if *file* operands were
113 specified, and the name of the file in the archive if they were not.

114 If the `-r` option is used with the `-v` option, and *file* is already in the archive, the
115 standard output format is:

116 `"r - %s\n", <file>`

117 where *file* is the operand specified on the command line.

118 If *file* is being added to the archive with the `-r` option, the standard output format
119 is:

120 `"a - %s\n", <file>`

121 where *file* is the operand specified on the command line.

122 If the `-t` option is used, `ar` writes the names of the files to the standard output in
123 the format:

124 `"%s\n", <file>`

125 where *file* is the operand specified on the command line, if *file* operands were
126 specified, or the name of the file in the archive if they were not.

127 If the `-t` option is used with the `-v` option, the standard output format is:

128 `"%s %u/%u %u %s %d %d:%d %d %s\n", <member mode>, <user ID>,
129 <group ID>, <number of bytes in member>, <abbreviated month>, <day-
130 of-month>, <hour>, <minute>, <year>, <file>`

131 Where:

132 *file* shall be the operand specified on the command line, if *file*
 133 operands were specified, or the name of the file in the
 134 archive if they were not.

135 *<member mode>* shall be formatted the same as the *<file mode>* string
 136 defined in 4.39.6.1 (Standard Output of *ls*), except that the
 137 first character, the *<entry type>*, is not used; the string
 138 represents the file mode of the archive member at the time it
 139 was added to, or replaced in, the archive.

140 The following represent the last-modification time of a file when it was most
 141 recently added to or replaced in the archive:

142 *<abbreviated month>*
 143 shall be equivalent to the *%b* format in date (see 4.15).

144 *<day-of-month>* shall be equivalent to the *%e* format in date.

145 *<hour>* shall be equivalent to the *%H* format in date.

146 *<minute>* shall be equivalent to the *%M* format in date.

147 *<year>* shall be equivalent to the *%Y* format in date.

148 When **LC_TIME** does not specify the POSIX Locale, a different format and order of
 149 presentation of these fields relative to each other may be used in a format
 150 appropriate in the specified locale.

151 If the *-x* option is used with the *-v* option, the standard output format is:

152 "x - %s\n", *<file>*

153 where *file* is the operand specified on the command line, if *file* operands were
 154 specified, or the name of the file in the archive if they were not.

155 **6.1.6.2 Standard Error**

156 Used only for diagnostic messages. The diagnostic message about creating a new
 157 archive when *-c* is not specified shall not modify the exit status.

158 **6.1.6.3 Output Files**

159 Archives are files with unspecified formats.

160 **6.1.7 Extended Description**

161 None.

6.1.8 Exit Status

The `ar` utility shall exit with one of the following values:

- 0 Successful completion.
- >0 An error occurred.

6.1.9 Consequences of Errors

Default.

6.1.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The archive format is not described. It is recognized that there are several known `ar` formats, which are not compatible. The `ar` utility is being included, however, to allow creation of archives that are intended for use only on the same machine. The archive file is specified as a file and it can be moved as a file. This does allow an archive to be moved from one machine to another machine that uses the same implementation of `ar`.

Utilities such as `pax` (and its forebears `tar` and `cpio`) also provide portable “archives.” This is a not a duplication; the `ar` interface is included in the standard to provide an interface primarily for `make` and the compilers, based on a historical model.

In historical implementations, the `-q` option is known to execute quickly because `ar` does not check whether the added members are already in the archive. This is useful to bypass the searching otherwise done when creating a large archive piece-by-piece. The remarks may or may not hold true for a brand-new POSIX.2 implementation; and hence, these remarks have been moved out of the specification and into the Rationale.

Likewise, historical implementations maintain a symbol table to speed searches, particularly when the archive contains object files. However, future implementors may or may not use a symbol table, and the `-s` option was removed from this clause to permit implementors freedom of choice. Instead, the requirement that archive libraries be suitable for link editing was added to ensure the intended functionality. Systems such as System V maintain the symbol table without requiring the use of `-s`, so adding `-s` (even if it were worded as allowing a no-op) would essentially require all portable applications to use it in all invocations involving libraries.

The Operands subclause requires what might seem to be true without specifying it: the archive cannot truncate the filenames below `{NAME_MAX}`. Some historical implementations do so, however, causing unexpected results for the application. Therefore, POSIX.2 makes the requirement explicit to avoid misunderstandings.

According to the System V documentation, the options `-dmpqrtx` are not required to begin with a hyphen (`-`). POSIX.2 requires that a conforming application use the leading hyphen.

When extracting files with long filenames into a file system that supports only shorter filenames, an undefined condition occurs. Typical implementation actions might be one of the following:

- Extract and truncate the filename only when an existing file would not be overlaid.
- Extract and truncate the filename and overlay an existing file only if some extension such as another command-line option were used to override this safety feature.
- Refuse to extract any files unless an extension overrode the default.

The archive format used by the 4.4BSD implementation is documented in the rationale as an example:

A file created by `ar` begins with the “magic” string “!`<arch>`\n”. The rest of the archive is made up of objects, each of which is composed of a header for a file, a possible filename, and the file contents. The header is portable between machine architectures, and, if the file contents are printable, the archive is itself printable.

The header is made up of six ASCII fields, followed by a two-character trailer. The fields are the object name (16 characters), the file last modification time (12 characters), the user and group IDs (each 6 characters), the file mode (8 characters) and the file size (10 characters). All numeric fields are in decimal, except for the file mode, which is in octal.

The modification time is the file `st_mtime` field. The user and group IDs are the file `st_uid` and `st_gid` fields. The file mode is the file `st_mode` field. The file size is the file `st_size` field. The two-byte trailer is the string “\n”.

Only the name field has any provision for overflow. If any filename is more than 16 characters in length or contains an embedded space, the string “#1/” followed by the ASCII length of the name is written in the name field. The file size (stored in the archive header) is incremented by the length of the name. The name is then written immediately following the archive header.

Any unused characters in any of these fields are written as `<space>` characters. If any fields are their particular maximum number of characters in length, there will be no separation between the fields.

Objects in the archive are always an even number of bytes long; files that are an odd number of bytes long are padded with a `<newline>` character, although the size in the header does not reflect this.

History of Decisions Made

The `ar` utility description requires that (when all its members are valid object files) `ar` produce an object code library, which the linkage editor can use to extract object modules. If the linkage editor needs a symbol table to permit random access to the archive, `ar` must provide it; however, `ar` does not require a symbol table. The historical `-m` and `-q` positioning options were omitted, as were the positioning modifiers formerly associated with the `-m` and `-r` options, because the two functions of positioning are handled by the `ranlib`-style (a utility found on some historical systems to create symbol tables within the archive) symbol tables and/or the ability of portable applications to create multiple archives instead of loading from a single archive.

Earlier drafts had elaborate descriptions in the Asynchronous Events subclause about how signals were caught and then resent to itself. These were removed in favor of the default case because they are essentially implementation details, unnecessary for the application. Similarly, information about where (and if) temporary files are created was removed from earlier drafts.

The BSD `-o` option was omitted. It is a rare portable application that will use `ar` to extract object code from a library with concern for its modification time, since this can only be of importance to make. Hence, since this functionality is not deemed important for applications portability, the modification time of the extracted files is set to the current time.

There is at least one known implementation (for a small computer) that can accommodate only object files for that system, disallowing mixed object and other files. The ability to handle any type of file is not only existing practice for most implementations, but is also a reasonable expectation.

Consideration was given to changing the output format of `ar -tv` to the same format as the output of `ls -l`. This would have made parsing the output of `ar` the same as that of `ls`. This was rejected in part because the current `ar` format is commonly used and changes would break existing usage. Second, `ar` gives the user ID and group ID in numeric format separated by a slash. Changing this to be the user name and group name would not be right if the archive were moved to a machine that contained a different user database. Since `ar` cannot know whether the archive file was generated on the same machine, it cannot tell what to report.

The text on the `-ur` option combination is historical practice—since one filename can easily represent two different files (e.g., `/a/foo` and `/b/foo`), it is reasonable to replace the member in the archive even when the modification time in the archive is identical to that in the file system.

6.2 make — Maintain, update, and regenerate groups of programs

6.2.1 Synopsis

```
make [-einpqrst] [-f makefile] ... [ -k | -S ] [macro=name] ...
      [target_name ... ]
```

6.2.2 Description

The `make` utility can be used as a part of software development to update files that are derived from other files. A typical case is one where object files are derived from the corresponding source files. The `make` utility examines time relationships and updates those derived files (called targets) that have modified times earlier than the modified times of the files (called prerequisites) from which they are derived. A description file (“makefile”) contains a description of the relationships between files, and the commands that must be executed to update the targets to reflect changes in their prerequisites. Each specification, or rule, shall consist of a target, optional prerequisites, and optional commands to be executed when a prerequisite is newer than the target. There are two types of rules:

- Inference rules, which have one target name with at least one period (.) and no slash (/)
- Target rules, which can have more than one target name

In addition, `make` shall have a collection of built-in macros and inference rules that infer prerequisite relationships to simplify maintenance of programs.

To receive exactly the behavior described in this clause, a portable makefile shall:

- Include the special target `.POSIX` (see 6.2.7.3)
- Omit any special target reserved for implementations (a leading period followed by uppercase letters) that has not been specified by this clause.

The behavior of `make` is unspecified if either or both of these conditions are not met.

6.2.3 Options

The `make` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

- `-e` Cause environment variables, including those with null values, to override macro assignments within makefiles.

309	<code>-f <i>makefile</i></code>	Specify a different makefile. The argument <i>makefile</i> is a path-	
310		name of a description file, which is also referred to as the	
311		<i>makefile</i> . A pathname of "-" shall denote the standard input.	
312		There can be multiple instances of this option, and they shall be	
313		processed in the order specified. The effect of specifying the same	
314		option-argument more than once is unspecified. See 6.2.7.1.	
315	<code>-i</code>	Ignore error codes returned by invoked commands. This mode is	
316		the same as if the special target <code>.IGNORE</code> were specified without	1
317		prerequisites. See 6.2.7.2.	1
318	<code>-k</code>	Continue to update other targets that do not depend on the	
319		current target if a nonignored error occurs while executing the	
320		commands to bring a target up to date.	
321	<code>-n</code>	Write commands that would be executed on standard output, but	
322		do not execute them. However, lines with a plus-sign (+) prefix	
323		shall be executed. In this mode, lines with an at-sign (@) charac-	
324		ter prefix shall be written to standard output.	
325	<code>-p</code>	Write to standard output the complete set of macro definitions	
326		and target descriptions. The output format is unspecified.	
327	<code>-q</code>	Return a zero exit value if the target file is up-to-date; otherwise,	
328		return an exit value of 1. Targets shall not be updated if this	
329		option is specified. However, a command line (associated with the	
330		targets) with a plus-sign (+) prefix shall be executed.	
331	<code>-r</code>	Clear the suffix list and do not use the built-in rules.	
332	<code>-S</code>	Terminate make if an error occurs while executing the commands	
333		to bring a target up-to-date. This shall be the default and the	
334		opposite of <code>-k</code> .	
335	<code>-s</code>	Do not write command lines or touch messages (see <code>-t</code>) to stan-	
336		dard output before executing. This mode shall be the same as if	
337		the special target <code>.SILENT</code> were specified without prerequisites.	1
338		See 6.2.7.2.	1
339	<code>-t</code>	Update the modification time of each target as though a touch	
340		<i>target</i> had been executed. See touch in 4.63. Targets that have	1
341		prerequisites but no commands (see 6.2.7.3), or that are already	1
342		up-to-date, shall not be touched in this manner. Write messages	1
343		to standard output for each target file indicating the name of the	
344		file and that it was touched. Normally, the command lines associ-	
345		ated with each target are not executed. However, a command line	
346		with a plus-sign (+) prefix shall be executed.	
347	If the <code>-k</code> and <code>-S</code> options are both specified on the command line, by the		
348	MAKEFLAGS environment variable, or by the MAKEFLAGS macro, the last one		
349	evaluated shall take precedence. The MAKEFLAGS environment variable shall		
350	be evaluated first and the command line shall be evaluated second. Assignments		
351	to the MAKEFLAGS macro shall be evaluated as described in 6.2.5.3.		

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

6.2.4 Operands

The following operands shall be supported by the implementation:

target_name Target names, as defined in 6.2.7. If no target is specified, while *make* is processing the makefiles, the first target that *make* encounters that is not a special target or an inference rule shall be used.

macro=name Macro definitions, as defined in 6.2.7.4.

If the *target_name* and *macro=name* operands are intermixed on the command line, the results are unspecified.

6.2.5 External Influences

6.2.5.1 Standard Input

The standard input shall be used only if the *makefile* option-argument is *-*. See Input Files.

6.2.5.2 Input Files

The input file, otherwise known as the makefile, is a text file containing rules, macro definitions, and comments. (See 6.2.7.)

6.2.5.3 Environment Variables

The following environment variables shall affect the execution of *make*:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).

LC_MESSAGES This variable shall determine the language in which messages should be written.

MAKEFLAGS

This variable shall be interpreted as a character string representing a series of option characters to be used as the default options. The implementation shall accept both of the following formats (but need not accept them when intermixed):

- (1) The characters are option letters without the leading hyphens or <blank> separation used on a command line.
- (2) The characters are formatted in a manner similar to a portion of the make command line: options are preceded by hyphens and <blank>-separated as described in 2.10.2. The *macro=name* macro definition operands can also be included. The difference between the contents of **MAKEFLAGS** and the command line is that the contents of the variable shall not be subjected to the word expansions (see 3.6) associated with parsing the command line values.

When the command-line options `-f` or `-p` are used, they shall take effect regardless of whether they also appear in **MAKEFLAGS**. If they otherwise appear in **MAKEFLAGS**, the result is undefined.

The **MAKEFLAGS** variable shall be accessed from the environment before the makefile is read. At that time, all of the options (except `-f` and `-p`) and command-line macros not already included in **MAKEFLAGS** shall be added to the **MAKEFLAGS** macro. The **MAKEFLAGS** macro shall be passed into the environment as an environment variable for all child processes. If the **MAKEFLAGS** macro is subsequently set by the makefile, it shall replace the **MAKEFLAGS** variable currently found in the environment.

The value of the **SHELL** environment variable shall not be used as a macro and shall not be modified by defining the **SHELL** macro in a makefile or on the command line. All other environment variables, including those with null values, shall be used as macros, as defined in 6.2.7.4.

6.2.5.4 Asynchronous Events

If not already ignored, make shall trap **SIGHUP**, **SIGTERM**, **SIGINT**, and **SIGQUIT** and remove the current target unless the target is a directory or the target is a prerequisite of the special target **.PRECIOUS** or unless one of the `-n`, `-p`, or `-q` options was specified. Any targets removed in this manner shall be reported in diagnostic messages of unspecified format, written to standard error. After this cleanup process, if any, make shall take the standard action for all other signals; see 2.11.5.4.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

6.2.6 External Effects

6.2.6.1 Standard Output

The `make` utility shall write all commands to be executed to standard output unless the `-s` option was specified, the command is prefixed with an at-sign, or the special target `.SILENT` has either the current target as a prerequisite or has no prerequisites. If `make` is invoked without any work needing to be done, it shall write a message to standard output indicating that no action was taken.

6.2.6.2 Standard Error

Used only for diagnostic messages.

6.2.6.3 Output Files

None. However, utilities invoked by `make` may create additional files.

6.2.7 Extended Description

The `make` utility attempts to perform the actions required to ensure that the specified target(s) are up-to-date. A target is considered out-of-date if it is older than any of its prerequisites or if it does not exist. The `make` utility shall treat all prerequisites as targets themselves and recursively ensure that they are up-to-date, processing them in the order in which they appear in the rule. The `make` utility shall use the modification times of files to determine if the corresponding targets are out-of-date. (See 2.9.1.6.)

After `make` has ensured that all of the prerequisites of a target are up-to-date, and if the target is out-of-date, the commands associated with the target entry shall be executed. If there are no commands listed for the target, the target shall be treated as up-to-date.

6.2.7.1 Makefile Syntax

A makefile can contain rules, macro definitions (see 6.2.7.4), and comments. There are two kinds of rules: inference rules (6.2.7.5) and target rules (6.2.7.3). The `make` utility shall contain a set of built-in inference rules. If the `-r` option is present, the built-in rules shall not be used and the suffix list shall be cleared. Additional rules of both types can be specified in a makefile. If a rule or macro is defined more than once, the value of the rule or macro shall be that of the last one specified. Comments start with a number-sign (#) and continue until an unescaped <newline> is reached.

By default, the file `./makefile` shall be used. If `./makefile` is not found, the file `./Makefile` shall be tried. If neither `./makefile` nor `./Makefile` are found, other implementation-defined pathnames may also be tried.

The `-f` option shall direct `make` to ignore `./makefile` and `./Makefile` (and any implementation-defined variants) and use the specified argument as a makefile instead. If the `-` argument is specified, standard input shall be used.

The term *makefile* is used to refer to any rules provided by the user whether in `./makefile`, `./Makefile`, or specified by the `-f` option.

The rules in makefiles shall consist of the following types of lines: target rules, including special targets (see 6.2.7.3); inference rules (see 6.2.7.5); macro definitions (see 6.2.7.4); empty lines; and comments. Comments start with a number sign (#) and continue until an unescaped `<newline>` is reached.

When an escaped `<newline>` (one preceded by a backslash) is found anywhere in the makefile, it shall be replaced, along with any leading white space on the following line, with a single `<space>`.

6.2.7.2 Makefile Execution

Command lines shall be processed one at a time by writing the command line to the standard output (unless one of the conditions listed below under “@” suppresses the writing) and executing the command(s) in the line. A `<tab>` character may precede the command to standard output. Commands shall be executed by passing the command line to the command interpreter in the same manner as if the string were the argument to the function in 7.1.1 [such as the `system()` function in the C binding].

The environment for the command being executed shall contain all of the variables in the environment of `make`. The macros from the command line to `make` shall be added to `make`’s environment. Other implementation-defined variables may also be added to `make`’s environment. If any command-line macro has been defined elsewhere, the command-line value shall overwrite the existing value. If the **MAKEFLAGS** variable is not set in the environment in which `make` was invoked, in the makefile, or on the command line, it shall be created by `make`, and shall contain all options specified on the command line except for the `-f` and `-p` options. It may also contain implementation-defined options.

By default, when `make` receives a nonzero status from the execution of a command, it terminates with an error message to standard error.

Command lines can have one or more of the following prefixes: a hyphen (`-`), an at-sign (`@`), or a plus-sign (`+`). These modify the way in which `make` processes the command. When a command is written to standard output, the prefix shall not be included in the output.

- If the command prefix contains a hyphen, or the `-i` option is present, or the special target `.IGNORE` has either the current target as a prerequisite or has no prerequisites, any error found while executing the command shall be ignored.
- @ If the command prefix contains an at-sign and the command-line `-n` option is not specified, or the `-s` option is present, or the special target `.SILENT` has either the current target as a prerequisite or has no prerequisites, the

505 command shall not be written to standard output before it is executed.

506 + If the command prefix contains a plus-sign, this indicates a command line

507 that shall be executed even if `-n`, `-q`, or `-t` is specified.

508 **6.2.7.3 Target Rules**

509 Target rules are formatted as follows:

```

510       target [target ...]: [prerequisite ...][;command]           1
511       [<tab>command                                               1
512       <tab>command                                               1
513       ...]                                                           1
514       (line that does not begin with <tab>)                     1

```

515 Target entries are specified by a <blank>-separated, nonnull list of targets, then

516 a colon, then a <blank>-separated, possibly empty list of prerequisites. Text fol-

517 lowing a semicolon, if any, and all following lines that begin with a <tab>, are 1

518 command lines to be executed to update the target. The first nonempty line that 1

519 does not begin with a <tab> or # shall begin a new entry. An empty or blank 1

520 line, or a line beginning with #, may begin a new entry. 1

521 Applications shall select target names from the set of characters consisting solely

522 of periods, underscores, digits, and alphabets from the portable character set

523 (see 2.4). Implementations may allow other characters in target names as exten-

524 sions. The interpretation of targets containing the characters “%” and “.” is 1

525 implementation defined. 1

526 A target that has prerequisites, but does not have any commands, can be used to

527 add to the prerequisite list for that target. Only one target rule for any given tar-

528 get can contain commands.

529 Lines that begin with one of the following are called *special targets* and control

530 the operation of `make`:

```

531       .DEFAULT   If the makefile uses this special target, it shall be specified with
532                   commands, but without prerequisites. The commands shall be
533                   used by make if there are no other rules available to build a tar-
534                   get.

535       .IGNORE    Prerequisites of this special target are targets themselves; this
536                   shall cause errors from commands associated with them to be
537                   ignored in the same manner as specified by the -i option. Subse-
538                   quent occurrences of .IGNORE shall add to the list of targets ignor-
539                   ing command errors. If no prerequisites are specified, make shall
540                   behave as if the -i option had been specified and errors from all
541                   commands associated with all targets shall be ignored.

542       .POSIX     This special target shall be specified without prerequisites or
543                   commands. If it appears before the first noncomment line in the
544                   makefile, make shall process the makefile as specified by this
545                   clause; otherwise, the behavior of make is unspecified.

```

546 .PRECIOUS Prerequisites of this special target shall not be removed if make
 547 receives one of the asynchronous events explicitly described in
 548 6.2.5.4. Subsequent occurrences of .PRECIOUS shall add to the
 549 list of precious files. If no prerequisites are specified, all targets
 550 in the makefile shall be treated as if specified with .PRECIOUS.

551 .SILENT Prerequisites of this special target are targets themselves; this
 552 shall cause commands associated with them to not be written to
 553 the standard output before they are executed. Subsequent
 554 occurrences of .SILENT shall add to the list of targets with silent
 555 commands. If no prerequisites are specified, make shall behave
 556 as if the -s option had been specified and no commands or touch
 557 messages associated with any target shall be written to standard
 558 output.

559 .SUFFIXES Prerequisites of .SUFFIXES shall be appended to the list of known
 560 suffixes and are used in conjunction with the inference rules (see
 561 6.2.7.5). If .SUFFIXES does not have any prerequisites, the list of
 562 known suffixes shall be cleared. Makefiles shall not associate
 563 commands with .SUFFIXES.

564 Targets with names consisting of a leading period followed by the uppercase
 565 letters POSIX and then any other characters are reserved for future standardiza-
 566 tion. Targets with names consisting of a leading period followed by one or more
 567 uppercase letters are reserved for implementation extensions.

568 6.2.7.4 Macros

569 Macro definitions are in the form:

570 *string1* = [*string2*] 1

571 The macro named *string1* is defined as having the value of *string2*, where *string2*
 572 is defined as all characters, if any, after the equals-sign, up to a comment charac- 1
 573 ter (#) or an unescaped <newline> character. Any <blank>s immediately before
 574 or after the equals-sign shall be ignored.

575 Subsequent appearances of \$(*string1*) or \${*string1*} shall be replaced by *string2*.
 576 The parentheses or braces are optional if *string1* is a single character. The macro
 577 \$\$ shall be replaced by the single character \$.

578 Applications shall select macro names from the set of characters consisting solely 2
 579 of periods, underscores, digits, and alphabets from the portable character set 2
 580 (see 2.4). A macro name shall not contain an equals-sign. Implementations may 2
 581 allow other characters in macro names as extensions. 2

582 Macros can appear anywhere in the makefile. Macros in target lines shall be
 583 evaluated when the target line is read. Macros in command lines shall be
 584 evaluated when the command is executed. Macros in macro definition lines shall
 585 not be evaluated until the new macro being defined is used in a rule or command.
 586 A macro that has not been defined shall evaluate to a null string without causing
 587 any error condition.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

The forms $\$(string1[:subst1=[subst2]])$ or $\$\{string1[:subst1=[subst2]]\}$ can be used to replace all occurrences of *subst1* with *subst2* when the macro substitution is performed. The *subst1* to be replaced shall be recognized when it is a suffix at the end of a word in *string1* (where a “word,” in this context, is defined to be a string delimited by the beginning of the line, a <blank>, or a <newline>).

Macro assignments shall be accepted from the sources listed below, in the order shown. If a macro name already exists at the time it is being processed, the newer definition shall replace the existing definition.

- (1) Macros defined in make’s built-in inference rules.
- (2) The contents of the environment, including the variables with null values, in the order defined in the environment.
- (3) Macros defined in the makefile(s), processed in the order specified.
- (4) Macros specified on the command line. It is unspecified whether the internal macros defined in 6.2.7.7 are accepted from the command line.

If the `-e` option is specified, the order of processing sources (2) and (3) shall be reversed.

The `SHELL` macro shall be treated specially. It shall be provided by make and set to the pathname of the shell command language interpreter (see `sh` in 4.56). The **SHELL** environment variable shall not affect the value of the `SHELL` macro. If `SHELL` is defined in the makefile or is specified on the command line, it shall replace the original value of the `SHELL` macro, but shall not affect the **SHELL** environment variable. Other effects of defining `SHELL` in the makefile or on the command line are implementation defined.

6.2.7.5 Inference Rules

Inference rules are formatted as follows:

```

target :
<tab>command
[<tab>command]
...
(line that does not begin with <tab> or #)
```

The *target* portion shall be a valid target name (see 6.2.7.3) and shall be of the form *.s2* or *.s1.s2* (where *.s1* and *.s2* are suffixes that have been given as prerequisites of the `.SUFFIXES` special target and *s1* and *s2* do not contain any slashes or periods.) If there is only one period in the target, it is a single-suffix inference rule. Targets with two periods are double-suffix inference rules. Inference rules can have only one target before the colon.

The makefile shall not specify prerequisites for inference rules; no characters other than white space shall follow the colon in the first line, except when creating the “empty rule,” described below. Prerequisites are inferred, as described below.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

Inference rules can be redefined. A target that matches an existing inference rule shall overwrite the old inference rule. An “empty rule” can be created with a command consisting of simply a semicolon (that is, the rule still exists and is found during inference rule search, but since it is empty, execution has no effect). The empty rule also can be formatted as follows:

```
rule: ;
```

where zero or more <blank>s separate the colon and semicolon.

The `make` utility uses the suffixes of targets and their prerequisites to infer how a target can be made up-to-date. A list of inference rules defines the commands to be executed. By default, `make` contains a built-in set of inference rules. Additional rules can be specified in the makefile.

The special target `.SUFFIXES` contains as its prerequisites a list of suffixes that are to be used by the inference rules. The order in which the suffixes are specified defines the order in which the inference rules for the suffixes are used. New suffixes shall be appended to the current list by specifying a `.SUFFIXES` special target in the makefile. A `.SUFFIXES` target with no prerequisites shall clear the list of suffixes. An empty `.SUFFIXES` target followed by a new `.SUFFIXES` list is required to change the order of the suffixes.

Normally, the user would provide an inference rule for each suffix. The inference rule to update a target with a suffix `.s1` from a prerequisite with a suffix `.s2` is specified as a target `.s2.s1`. The internal macros provide the means to specify general inference rules. (See 6.2.7.7.)

When no target rule is found to update a target, the inference rules shall be checked. The suffix of the target (`.s1`) to be built is compared to the list of suffixes specified by the `.SUFFIXES` special targets. If the `.s1` suffix is found in `.SUFFIXES`, the inference rules shall be searched in the order defined for the first `.s2.s1` rule whose prerequisite file (`$*.s2`) exists. If the target is out-of-date with respect to this prerequisite, the commands for that inference rule shall be executed.

If the target to be built does not contain a suffix and there is no rule for the target, the single suffix inference rules shall be checked. The single-suffix inference rules define how to build a target if a file is found with a name that matches the target name with one of the single suffixes appended. A rule with one suffix `.s2` is the definition of how to build `target` from `target.s2`. The other suffix (`.s1`) is treated as null.

6.2.7.6 Libraries

If a target or prerequisite contains parentheses, it shall be treated as a member of an archive library. For the `lib(member.o)` expression `lib` refers to the name of the archive library and `member.o` to the member name. The member shall be an object file with the `.o` suffix. The modification time of the expression is the modification time for the member as kept in the archive library. See 6.1. The `.a` suffix refers to an archive library. The `.s2.a` rule is used to update a member in the library from a file with a suffix `.s2`.

6.2.7.7 Internal Macros

The `make` utility shall maintain five internal macros that can be used in target and inference rules. In order to clearly define the meaning of these macros, some clarification of the terms “target rule,” “inference rule,” “target,” and “prerequisite” is necessary.

Target rules are specified by the user in a makefile for a particular target. Inference rules are user- or make-specified rules for a particular class of target names. Explicit prerequisites are those prerequisites specified in a makefile on target lines. Implicit prerequisites are those prerequisites that are generated when inference rules are used. Inference rules are applied to implicit prerequisites or to explicit prerequisites that do not have target rules defined for them in the makefile. Target rules are applied to targets specified in the makefile.

Before any target in the makefile is updated, each of its prerequisites (both explicit and implicit) shall be updated. This shall be accomplished by recursively processing each prerequisite. Upon recursion, each prerequisite shall become a target itself. Its prerequisites in turn shall be processed recursively until a target is found that has no prerequisites, at which point the recursion shall stop. The recursion then shall back up, updating each target as it goes.

In the definitions that follow, the word “target” refers to one of:

- A target specified in the makefile,
- An explicit prerequisite specified in the makefile that becomes the target when `make` processes it during recursion, or
- An implicit prerequisite that becomes a target when `make` processes it during recursion.

In the definitions that follow, the word “prerequisite” refers to either:

- An explicit prerequisite specified in the makefile for a particular target, or
- An implicit prerequisite generated as a result of locating an appropriate inference rule and corresponding file that matches the suffix of the target.

The five internal macros are:

`$$` The `$$` macro shall evaluate to the full target name of the current target, or the archive filename part of a library archive target. It shall be evaluated for both target and inference rules.

For example, in the `.c.a` inference rule, `$$` represents the out-of-date `.a` file to be built. Similarly, in a makefile target rule to build `lib.a` from `file.c`, `$$` represents the out-of-date `lib.a`.

`$(libname)` The `$(libname)` macro shall be evaluated only when the current target is an archive library member of the form `libname(member.o)`. In these cases, `$$` shall evaluate to `libname` and `$(libname)` shall evaluate to `member.o`. The `$(libname)` macro shall be evaluated for both target and inference rules.

709 For example, in a makefile target rule to build `lib.a(file.o)`, `$$` 1
710 represents `file.o`—as opposed to `$$`, which represents `lib.a.` 1

711 `$$` The `$$` macro shall evaluate to the list of prerequisites that are newer 1
712 than the current target. It shall be evaluated for both target and infer- 1
713 ence rules. 1

714 For example, in a makefile target rule to build `prog` from `file1.o`, 1
715 `file2.o`, and `file3.o`, and where `prog` is not out of date with respect 1
716 to `file1.o`, but is out of date with respect to `file2.o` and `file3.o`, 1
717 `$$` represents `file2.o` and `file3.o`. 1

718 `$$<` In an inference rule, `$$<` shall evaluate to the file name whose existence 1
719 allowed the inference rule to be chosen for the target. In the `.DEFAULT` 1
720 rule, the `$$<` macro shall evaluate to the current target name. The `$$<` 1
721 macro shall be evaluated only for inference rules. 1

722 For example, in the `.c.a` inference rule, `$$<` represents the prerequisite 1
723 `.c` file. 1

724 `$$*` The `$$*` macro shall evaluate to the current target name with its suffix 1
725 deleted. It shall be evaluated at least for inference rules. 2

726 For example, in the `.c.a` inference rule, `$$*.o` represents the out-of-date 1
727 `.o` file that corresponds to the prerequisite `.c` file. 1

728 Each of the internal macros has an alternate form. When an uppercase D or F is
729 appended to any of the macros, the meaning is changed to the *directory part* for D
730 and *filename part* for F. The directory part is the path prefix of the file without a
731 trailing slash; for the current directory, the directory part is `."`. When the `$$?`
732 macro contains more than one prerequisite filename, the `$(?D)` and `$(?F)` [or
733 `${?D}` and `${?F}`] macros expand to a list of directory name parts and filename
734 parts respectively.

735 For the target `lib(member.o)` and the `.s2.a` rule, the internal macros are defined
736 as:

737 `$$<` `member.s2`
738 `$$*` `member`
739 `$$@` `lib`
740 `$$?` `member.s2`
741 `$$%` `member.o`

742 6.2.7.8 Default Rules

743 The default rules for `make` shall achieve results that are the same as if the follow-
744 ing were used. Implementations that do not support the C Language Develop-
745 ment Utilities Option may omit `CC`, `CFLAGS`, `YACC`, `YFLAGS`, `LEX`, `LFLAGS`, `LDFLAGS`,
746 and the `.c`, `.y`, and `.l` inference rules. Implementations that do not support the
747 FORTRAN Language Development Utilities Option may omit `FC`, `FFLAGS`, and the
748 `.f` inference rules. Implementations may provide additional macros and rules.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

749 NOTE: In a future version of this standard, the default rules may be specified separately from the
 750 make clause, such as with the language-dependent development options.

751 *SUFFIXES AND MACROS*

752 .SUFFIXES: .o .c .y .l .a .sh .f 1

753 MAKE=make

754 AR=ar

755 ARFLAGS=-rv

756 YACC=yacc

757 YFLAGS=

758 LEX=lex

759 LFLAGS=

760 LDFLAGS=

761 CC=c89

762 CFLAGS=-O

763 FC=fort77

764 FFLAGS=-O 1

765 *SINGLE SUFFIX RULES*

766 .c:

767 \$(CC) \$(CFLAGS) \$(LDFLAGS) -o \$@ \$<

768 .f:

769 \$(FC) \$(FFLAGS) \$(LDFLAGS) -o \$@ \$<

770 .sh:

771 cp \$< \$@

772 chmod a+x \$@

773 *DOUBLE SUFFIX RULES*

774 .c.o:

775 \$(CC) \$(CFLAGS) -c \$<

776 .f.o:

777 \$(FC) \$(FFLAGS) -c \$<

778 .y.o:

779 \$(YACC) \$(YFLAGS) \$<

780 \$(CC) \$(CFLAGS) -c y.tab.c

781 rm -f y.tab.c 1

782 mv y.tab.o \$@

783 .l.o:

784 \$(LEX) \$(LFLAGS) \$<

785 \$(CC) \$(CFLAGS) -c lex.yy.c

786 rm -f lex.yy.c 1

787 mv lex.yy.o \$@

788 .y.c:

789 \$(YACC) \$(YFLAGS) \$<

790 mv y.tab.c \$@

791 .l.c:

792 \$(LEX) \$(LFLAGS) \$<

793 mv lex.yy.c \$@

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

```

794      .c.o:
795          $(CC) -c $(CFLAGS) $<
796          $(AR) $(ARFLAGS) $@ $*.o
797          rm -f $*.o
798
799      .f.o:
800          $(FC) -c $(FFLAGS) $<
801          $(AR) $(ARFLAGS) $@ $*.o
802          rm -f $*.o

```

6.2.8 Exit Status

When the `-q` option is specified, the `make` utility shall exit with one of the following values:

- 0 Successful completion.
- 1 The target was not up-to-date.
- >1 An error occurred.

When the `-q` option is not specified, the `make` utility shall exit with one of the following values:

- 0 Successful completion.
- >0 An error occurred.

6.2.9 Consequences of Errors

Default.

6.2.10 Rationale. *(This subclause is not a part of P1003.2)*

The `make` provided here is intended to provide the means for changing portable source code into runnable executables on a POSIX.2 system. It reflects the most common features present in System V and BSD makes.

Historically, the `make` utility has been an especially fertile ground for vendor- and research-organization-specific syntax modifications and extensions. Examples include:

- Syntax supporting parallel execution (Sequent, Cray, GNU, and others)
- Additional “operators” separating targets and their prerequisites (System V, BSD, and others)
- Specifying that command lines containing the strings `${MAKE}` and `$(MAKE)` are executed when the `-n` option is specified (GNU and System V)
- Modifications of the meaning of internal macros when referencing libraries (BSD and others)

- Using a single instance of the shell for all of a target’s command lines (BSD and others)
- Allowing spaces as well as tabs to delimit command lines (BSD)
- Adding C-preprocessor-style “include” and “ifdef” constructs (System V, GNU, BSD, and others)
- Remote execution of command lines (Sprite and others)
- Specifying additional special targets (Sun, BSD, System V, and most others).

Additionally, many vendors and research organizations have rethought the basic concepts of `make`, creating vastly extended, as well as completely new, syntaxes. Each of these versions of “`make`” fulfills the needs of a different community of users; it is unreasonable for this standard to require behavior that would be incompatible (and probably inferior) to existing practice for such a community.

In similar circumstances, when the industry has enough sufficiently incompatible formats as to make them irreconcilable, POSIX.2 has followed one or both of two courses of action. Commands have been renamed (`cksum`, `echo`, and `pax`) and/or command-line options have been provided to select the desired behavior (`grep`, `od`, and `pax`).

Because the syntax specified for the `make` utility is, by and large, a subset of the syntaxes accepted by almost all versions of `make`, it was decided that it would be counter-productive to change the name. And since the `makefile` itself is a basic unit of portability, it would not be completely effective to reserve a new option letter, such as `make -P`, to achieve the portable behavior. Therefore, the special target `.POSIX` was added to the `makefile`, allowing users to specify “standard” behavior. This special target does not preclude extensions in the `make` utility, or such extensions being used by the `makefile` specifying the target; it does, however, preclude any extensions from being applied that could alter the behavior of previously valid syntax; such extensions must be controlled via command-line options or new special targets. It is incumbent upon portable `makefiles` to specify the `.POSIX` special target in order to guarantee that they are not affected by local extensions.

The portable version of `make` described in this clause is not intended to be the state of the art software generation tool and, as such, some newer and more leading-edge features have not been included. An attempt has been made to describe the portable `makefile` in a manner that does not preclude such extensions as long as they do not disturb the portable behavior described here.

One use of this `make` and the `makefile` syntax is as a format that newer versions of `make` can generate for portability purposes.

Examples, Usage

The following command:

```
make
```

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

makes the first target found in the makefile.

The following command:

```
make junk
```

makes the target `junk`.

The following makefile says that `pgm` depends on two files, `a.o` and `b.o`, and that they in turn depend on their corresponding source files (`a.c` and `b.c`), and a common file `incl.h`:

```
pgm: a.o b.o
    c89 a.o b.o -o pgm

a.o: incl.h a.c
    c89 -c a.c

b.o: incl.h b.c
    c89 -c b.c
```

An example for making optimized `.o` files from `.c` files is:

```
.c.o:
    c89 -c -O $*.c
```

or:

```
.c.o:
    c89 -c -O $<
```

The most common use of the archive interface follows. Here, it is assumed that the source files are all C language source:

```
lib: lib(file1.o) lib(file2.o) lib(file3.o)
    @echo lib is now up-to-date
```

The `.c.a` rule is used to make `file1.o`, `file2.o`, and `file3.o` and insert them into `lib`. 1
1

The `-k` and `-S` options are both present so that the relationship between the command line, the **MAKEFLAGS** variable, and the makefile can be controlled precisely. If the `k` flag is passed in **MAKEFLAGS** and a command is of the form:

```
$(MAKE) -S foo
```

then the default behavior is restored for the child `make`.

When the `-n` option is specified, it is always added to **MAKEFLAGS**. This allows a recursive `make -n target` to be used to see all of the action that would be taken to update *target*.

The definition of **MAKEFLAGS** allows both the System V letter string and the BSD command-line formats. The two formats are sufficiently different to allow implementations to support both without ambiguity.

Because of widespread historical practice, interpreting a `#` number sign inside a variable as the start of a comment has the unfortunate side effect of making it impossible to place a number sign in a variable, thus forbidding something like

```

908         CFLAGS = "-D COMMENT_CHAR=' #' "
909
910 Earlier drafts stated that an “unquoted” number sign was treated as the start of a
911 comment. The make utility does not pay any attention to quotes. A number sign
912 starts a comment regardless of its surroundings.
913
914 The treatment of escaped <newline>s throughout the makefile is historical prac-
915 tice. For example, the inference rule:
916
917         .c.o\
918         :
919
920 works and the macro
921
922         f=      bar baz\
923
924         biz
925
926         a:
927         echo ==$f==
928
929 will echo ==bar baz biz==.
930
931 If $? were
932
933         /usr/include/stdio.h /usr/include/unistd.h foo.h
934
935 then $(?D) would be
936
937         /usr/include /usr/include .
938
939 and $(?F) would be
940
941         stdio.h unistd.h foo.h
942
943 The contents of the built-in rules can be viewed by running:
944
945         make -p -f /dev/null 2>/dev/null
946
947 Many historical makes stop chaining together inference rules when an intermedi- 1
948 ate target is nonexistent. For example, it might be possible for a make to deter- 1
949 mine that both .y.c and .c.o could be used to convert a .y to a .o. Instead, in 1
950 this case, make requires the use of a .y.o rule. 1
951
952 The text about “other implementation-defined pathnames may also be tried” in
953 addition to ./makefile and ./Makefile is to allow such extensions as
954 SCCS/s.Makefile and other variations. It was made an implementation-defined
955 requirement (as opposed to unspecified behavior) to highlight surprising imple-
956 mentations that might select something unexpected like /etc/Makefile.
957
958 For inference rules, the description of $< and $? seem similar. However, an
959 example shows the minor difference. In a makefile containing
960
961         foo.o: foo.h
962
963 if foo.h is newer than foo.o, yet foo.c is older than foo.o, the built-in rule to
964 make foo.o from foo.c will be used, with $< equal to foo.c and $? equal to
965 foo.h. (If foo.c is also newer than foo.o, $< is equal to foo.c and $? is equal
966 to “foo.h foo.c”.)

```

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

History of Decisions Made

Earlier drafts contained the macro `NPROC` as a means of specifying that `make` should use n processes to do the work required. While this feature is a valuable extension for many systems, it is not common usage and could require other non-trivial extensions to makefile syntax. This extension is not required by the standard, but could be provided as a compatible extension. The macro `PARALLEL` is used by some historical systems with essentially the same meaning (but without using a name that is a common system limit value). It is suggested that implementors recognize the existing use of `NPROC` and/or `PARALLEL` as extensions to `make`.

The default rules are based on System V. The default `CC=` value is `c89` instead of `cc` because POSIX.2 does not standardize the utility named `cc`. Thus, every conforming application would be required to define `CC=c89` to expect to run. There is no advantage conferred by the hope that the makefile might hit the “preferred” compiler because there is no way that this can be guaranteed to work. Also, since the portable makescript can only use the `c89` options, no advantage is conferred in terms of what the script can do. It is a quality of implementation issue as to whether `c89` is as good as `cc`.

Since SCCS and RCS are not part of POSIX.2, all `make` references to SCCS extensions have been omitted.

The `-d` option to `make` is frequently used to produce debugging information, but is too implementation-dependent to add to the standard.

The `-p` option is not passed in **MAKEFLAGS** on most existing implementations and to change this would cause many implementations to break without sufficiently increased portability.

Commands that begin with a plus-sign (+) are executed even if the `-n` option is present. Based on the GNU version of `make`, the behavior of `-n` when the plus-sign prefix is encountered has been extended to apply to `-q` and `-t` as well. However, the System V convention of forcing command execution with `-n` when a target’s command line contains either of the strings `$(MAKE)` or `${MAKE}` has not been adopted. This functionality appeared in earlier drafts, but the danger of this approach was pointed out with the following example of a portion of a makefile:

```
subdir:
    cd subdir; rm all_the_files; $(MAKE)
```

The loss of the System V behavior in this case is well-balanced by the safety afforded to other makefiles that were not aware of this situation. In any event, the command-line plus-sign prefix can provide the desired functionality.

The double colon in the target rule format is supported in BSD systems to allow more than one target line containing the same target name to have commands associated with it. Since this is not functionality described in the *SVID* or *XPG3*, it has been allowed as an extension, but not mandated.

The default rules are provided with text specifying that the built-in rules are to be the same *as if* the listed set were used. The intent is that implementations

should be able to use the rules without change, but will be allowed to alter them in ways that do not affect the primary behavior.

The best way to provide portable makefiles is to include all of the rules needed in the makefile itself. The rules provided use only features provided by other parts of the standard. The default rules include rules for optional commands in the standard. Only rules pertaining to commands that are provided are needed in an implementation's default set.

The argument could be made to drop the default rules list from the standard. They provide convenience, but do not enhance portability of applications. The prime benefit is in portability of users who wish to type `make command` and have the command build from a `command.c` file.

The historical **MAKESHELL** feature was omitted. In some implementations it is used to provide a way of letting a user override the shell to be used to run `make` commands. This was confusing; for a portable `make`, the shell should be chosen by the makefile writer or specified on the `make` command line and not by a user running `make`.

The `make` utilities in most historical implementations process the prerequisites of a target in left-to-right order, and the POSIX.2 makefile format requires this. It supports the standard idiom used in many makefiles that produce `yacc` programs, for example:

```
foo:    y.tab.o lex.o main.o
        $(CC) $(CFLAGS) -o $@ t.tab.o lex.o main.o
```

In this example, if `make` chose any arbitrary order, the `lex.o` might not be made with the correct `y.tab.h`. Although there may be better ways to express this relationship, it is widely used historically. Implementations that desire to update prerequisites in parallel should require an explicit extension to `make` or the makefile format to accomplish it, as described previously.

The algorithm for determining a new entry for target rules is partially unspecified. Some historical `makes` allow blank, empty, or comment lines within the collection of commands marked by leading `<tab>s`. A conforming makefile must ensure that each command starts with a `<tab>`, but implementations are free to ignore blank, empty, and comment lines without triggering the start of a new entry.

The Asynchronous Events subclause includes having `SIGTERM` and `SIGHUP`, along with the more traditional `SIGINT` and `SIGQUIT`, remove the current target unless directed not to. `SIGTERM` and `SIGHUP` were added to parallel other utilities that have historically cleaned up their work as a result of these signals. All but `SIGQUIT` is required to resend itself the signal it received to cause `make` to exit with a status that reflects the signal. The results from `SIGQUIT` are partially unspecified because, on systems that create `core` files upon receipt of `SIGQUIT`, the `core` from `make` would conflict with a `core` file from the command that was running when the `SIGQUIT` arrived. The main concern here was to prevent damaged files from appearing up-to-date when `make` is rerun.

The `.PRECIOUS` special target was extended to globally affect all targets (by specifying no prerequisites). The `.IGNORE` and `.SILENT` special targets were extended to allow prerequisites; it was judged to be more useful in some cases to be able to turn off errors or echoing for a list of targets than for the entire makefile. These extensions to System V's make were made to match historical practice from the BSD make.

Macros are not exported to the environment of commands to be run. This was never the case in any historical make and would have serious consequences. The environment is the same as the environment to make except that **MAKEFLAGS** and macros defined on the make command line are added.

Some implementations do not use `system()` for all command lines, as required by the POSIX.2 portable makefile format; as a performance enhancement, they select lines without shell metacharacters for direct execution by `execve()`. There is no requirement that `system()` be used specifically, but merely that the same results be achieved. The metacharacters typically used to bypass the direct `execve()` execution have been any of:

= | ^ () ; & < > * ? [] : \$ \ ' " \ \n

The default in some advanced versions of make is to group all the command lines for a target and execute them using a single shell invocation; the System V method is to pass each line individually to a separate shell. The single-shell method has the advantages in performance and the lack of a requirement for many continued lines. However, converting to this newer method has caused portability problems with many historical makefiles, so the behavior with the POSIX makefile is specified to be the same as System V's. It is suggested that the special target `.ONESHELL` be used as an implementation extension to achieve the single-shell grouping for a target or group of targets.

Novice users of make have had difficulty with the historical need to start commands with a `<tab>` character. Since it is often difficult to discern differences between `<tab>` and `<space>` characters on terminals or printed listings, confusing bugs can arise. In earlier drafts, an attempt was made to correct this problem by allowing leading `<blank>`s instead of `<tab>`s. However, implementors reported many makefiles that failed in subtle ways following this change and it is difficult to implement a make that unambiguously can differentiate between macro and command lines. There is extensive historical practice of allowing leading spaces before macro definitions. Forcing macro lines into column 1 would be a significant backward compatibility problem for some makefiles. Therefore, historical practice was restored.

The System V `INCLUDE` feature was considered, but not included. This would treat a line that began in the first column and contained `INCLUDE <filename>` as an indication to read `<filename>` at that point in the makefile. This is difficult to use in a portable way and it raises concerns about nesting levels and diagnostics. System V, BSD, GNU, and others have used different methods for including files.

Macros used within other macros are evaluated when the new macro is used rather than when the new macro is defined. Therefore:

```

1076     MACRO = value1
1077     NEW   = $(MACRO)
1078     MACRO = value2
1079     target:
1080         echo $(NEW)

```

1081 would produce *value2* and not *value1* since `NEW` was not expanded until it was
 1082 needed in the `echo` command line.

1083 The System V dynamic dependency feature was not added. It would support:

```

1084     cat: $$@.c

```

1085 that would expand to

```

1086     cat: cat.c

```

1087 This feature exists only in the new version of System V `make` and, while useful, is
 1088 not in wide usage. This means that macros are expanded twice for prerequisites:
 1089 once at makefile parse time and once at target update time.

1090 Consideration was given to adding metarules to the POSIX `make`. This would
 1091 make "`% .o: %.c`" the same as "`.c.o:`". This is quite useful and available from
 1092 some vendors, but it would cause too many changes to this `make` to support. It
 1093 would have introduced rule chaining and new substitution rules. However, the 1
 1094 rules for target names have been set to reserve the `%` and `"` characters. These are 1
 1095 traditionally used to implement metarules and quoting of target names, respec- 1
 1096 tively. Implementors are strongly encouraged to use these characters only for 1
 1097 these purposes. 1

1098 A request was made to extend the suffix delimiter character from a period to any
 1099 character. The metarules in newer `makes` solves this problem in a more general
 1100 way. POSIX.2 is staying with the more conservative historical definition until a
 1101 clear industry consensus on `make` technology might prompt a revision of this stan-
 1102 dard.

1103 The standard output format for the `-p` option is not described because it is pri-
 1104 marily a debugging option and the format is not generally useful to programs. In
 1105 historical implementations the output is not suitable for use in generating
 1106 makefiles. The `-p` format has been variable across historical implementations.
 1107 Therefore, the definition of `-p` was only to provide a consistently named option for
 1108 obtaining `make` script debugging information.

1109 Some historical implementations have not cleared the suffix list with `-r`.

1110 Implementations should be aware that some historical applications have inter-
 1111 mixed *target_name* and *macro=name* operands on the command line, expecting
 1112 that all of the macros will be processed before any of the targets are dealt with.
 1113 Portable applications do not do this, but some backward compatibility support
 1114 may be warranted.

1115 Empty inference rules are specified with a semicolon command rather than omit-
 1116 ting all commands, as described in a previous draft. The latter case has no tradi-
 1117 tional meaning and is reserved for implementation extensions, such as in GNU

1118 make.

1119 **6.3 strip — Remove unnecessary information from executable** 1120 **files**

1121 **6.3.1 Synopsis**

1122 strip *file*...

1123 **6.3.2 Description**

1124 The strip utility shall remove from executable files named by the *file* operands
1125 any information the implementor deems unnecessary to proper execution of those
1126 files. The nature of that information is unspecified. The effect of strip shall be
1127 the same as the use of the -s option to any of the compilers defined by this stan-
1128 dard.

1129 **6.3.3 Options**

1130 None.

1131 **6.3.4 Operands**

1132 The following operand shall be supported by the implementation:

1133 *file* A pathname referring to an executable file.

1134 **6.3.5 External Influences**

1135 **6.3.5.1 Standard Input**

1136 None.

1137 **6.3.5.2 Input Files**

1138 The input files shall be in the form of executable files successfully produced by
1139 any compiler defined by this standard.

1140 **6.3.5.3 Environment Variables**

1141 The following environment variables shall affect the execution of strip:

1142	LANG	This variable shall determine the locale to use for the
1143		locale categories when both LC_ALL and the correspond-
1144		ing environment variable (beginning with LC_) do not
1145		specify a locale. See 2.6.
1146	LC_ALL	This variable shall determine the locale to be used to over-
1147		ride any values for locale categories specified by the set-
1148		tings of LANG or any environment variables beginning
1149		with LC_ .
1150	LC_CTYPE	This variable shall determine the locale for the interpreta-
1151		tion of sequences of bytes of text data as characters (e.g.,
1152		single- versus multibyte characters in arguments).
1153	LC_MESSAGES	This variable shall determine the language in which mes-
1154		sages should be written.

1155 **6.3.5.4 Asynchronous Events**

1156 Default.

1157 **6.3.6 External Effects**

1158 **6.3.6.1 Standard Output**

1159 None.

1160 **6.3.6.2 Standard Error**

1161 Used only for diagnostic messages.

1162 **6.3.6.3 Output Files**

1163 The `strip` utility shall produce executable files of unspecified format.

1164 **6.3.7 Extended Description**

1165 None.

1166 **6.3.8 Exit Status**

1167 The `strip` utility shall exit with one of the following values:

1168 0 Successful completion.

1169 >0 An error occurred.

1170 **6.3.9 Consequences of Errors**

1171 Default.

1172 **6.3.10 Rationale.** *(This subclause is not a part of P1003.2)*

1173 **Examples, Usage**

1174 None.

1175 **History of Decisions Made**

1176 Historically, this utility has been used to remove the symbol table from an execut-
1177 able file. It was included since it is known that the amount of symbolic informa-
1178 tion can amount to several megabytes; the ability to remove it in a portable
1179 manner was deemed important, especially for smaller systems.

1180 The behavior of `strip` is said to be the same as the `-s` option to a compiler.
1181 While the end result is essentially the same it is not required to be identical. The
1182 same effect can be achieved with either `-s` during a compile or a `strip` on the
1183 final object file.

Section 7: Language-Independent System Services

This clause contains functional specifications for services that give applications access to features defined elsewhere in this standard. These services allow applications written in high-level languages to

- (1) execute commands using the shell language,
- (2) obtain values of environment variables,
- (3) perform regular expression and pattern matching,
- (4) process command arguments in a standard manner,
- (5) generate pathnames from a pattern,
- (6) perform shell word expansions,
- (7) obtain system configuration information, and
- (8) set locale control information

This clause does not define interfaces, but services that shall be provided by the interfaces in a language-dependent binding. This clause is optional, in that an implementation is not required to support any language binding to these services. However, any language binding shall support all of the services described here. Implementations therefore provide support for services in this clause by supplying a language-dependent binding such as the one defined in Annex B. Such a system would specify conformance to the language-dependent binding, not to the language-independent bindings given here.

7.0.1 Language-Independent System Services Rationale. *(This subclause is not a part of P1003.2)*

Section 7 essentially is a metastandard, in that it specifies services that must be in a language-dependent binding. An implementation conforms to a specific language-dependent binding such as for the C language, in Annex B, and the language-dependent binding must conform to the specifications in this clause.

In this standard, the language-independent specifications have not yet been developed. The language-independent syntax is being created in parallel by the POSIX.1 working group. Therefore, the C language bindings temporarily described in Annex B are actually the full interface specifications. It is the intention of the P1003.2 working group to rectify this situation in a later supplement by moving the majority of the interface specifications back into this clause,

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

leaving Annex B with only brief descriptions of the C bindings to those services.

This clause does not attempt to include everything that would be required of a language binding. The services here are those that are necessary to make use of features defined elsewhere in the standard, but that are not normally available in every language. Clearly a language that could not open, read, and write the files manipulated by the utilities in this standard would not be very useful, but this service is normally provided by any language and therefore isn't called out here. The ability to obtain values of environment variables exported from the shell, on the other hand, is not universally available, so that service is included here.

7.1 Shell Command Interface

7.1.1 Execute Shell Command

Any language binding to Language-Independent System Services shall include a facility to execute a shell command.

The language-independent specification for this facility has not been developed. The C binding for this facility is the *system()* function described in B.3.1.

7.1.2 Pipe Communications with Programs

Any language binding to Language-Independent System Services shall include a facility to execute a shell command, and to write the standard input or read the standard output of that command via a pipe.

The language-independent specification for this facility has not been developed. The C binding for this facility is the *popen()* and *pclose()* functions described in B.3.2.

7.2 Access Environment Variables

Any language binding to Language-Independent System Services shall include a facility to obtain values of environment variables, as specified in POSIX.1 {8}.

The language-independent specification for this facility has not been developed. The C binding for this facility is the *getenv()* function described in POSIX.1 {8} 4.6.1.

7.2.1 Access Environment Variables Rationale. *(This subclause is not a part of P1003.2)*

This facility is required in POSIX.2 so that applications can obtain values of exported shell variables.

7.3 Regular Expression Matching

Any language binding to Language-Independent System Services shall include a facility to interpret regular expressions as described in 2.8.

The language-independent specification for this facility has not been developed. The C binding is the *regcomp()*, *regexexec()*, and *regfree()* functions described in B.5.

7.3.1 Regular Expression Matching Rationale. *(This subclause is not a part of P1003.2)*

This service is important enough that it should be required by any language binding to POSIX.2.

Regular expression parsing and pattern matching are listed separately, since they are different services. A language binding could provide different functions to support regular expressions and patterns, or could combine them into a single function.

7.4 Pattern Matching

Any language binding to Language-Independent System Services shall include a facility to interpret patterns as described in 3.13.1 and 3.13.2. This facility shall allow the application to specify whether a slash character in the string to be matched will be treated as a regular character, or must be explicitly matched against a slash in the pattern.

The language-independent specification for this facility has not been developed. The C binding is the *fnmatch()* function described in B.6.

7.5 Command Option Parsing

Any language binding to Language-Independent System Services shall include a facility to parse the options and operands from the command line that invoked the application.

The language-independent specification for this facility has not been developed. The C binding for this facility is the *getopt()* function described in B.7.

7.6 Generate Pathnames Matching a Pattern

Any language binding to Language-Independent System Services shall include a facility to generate pathnames matching a pattern as described in 3.13.

The language-independent specifications for this facility has not been developed. The C binding is the *glob()* and *globfree()* functions described in B.8.

7.7 Perform Word Expansions

Any language binding to Language-Independent System Services shall include a facility to do shell word expansions as described in 3.6.

The language-independent specification for this facility has not been developed. The C binding is the *wordexp()* and *wordfree()* functions described in B.9.

7.7.1 Perform Word Expansions Rationale. *(This subclause is not a part of P1003.2)*

See the rationale for this function in B.9.

7.8 Get POSIX Configurable Variables

7.8.1 Get String-Valued Configurable Variables

Any language binding to Language-Independent System Services shall include a facility to obtain string configurable variables.

The language-independent specification for this facility has not been developed. The C binding for this facility is the *confstr()* function described in B.10.1.

7.8.2 Get Numeric-Valued Configurable Variables

Any language binding to Language-Independent System Services shall include facilities to determine the current values of system and pathname limits or options (*variables*), as specified by POSIX.1 {8}. The configurable variables listed in Table 7-1, which are defined in POSIX.1 {8}, shall be available in any POSIX.2 language-dependent binding, with minimum values as given in POSIX.1 {8}. Other POSIX.1 {8} configurable variables may be supported, but are not required by POSIX.2. This facility shall also make available current values for all system limits defined in 2.13.

The language-independent specifications for these facilities have not been developed. The C bindings are the *sysconf()* function described in POSIX.1 {8} 4.8, and the *pathconf()* and *fpathconf()* functions defined in POSIX.1 {8} 5.7.

7.8.2.1 Get Numeric-Valued Configurable Variables Rationale. *(This subclause is not a part of P1003.2)*

This description calls out specific values that `sysconf()`, `pathconf()`, and `fpathconf()` are required to support. Some of the POSIX.1 {8} values are excluded from this list because they are not relevant in a POSIX.2-only environment. Currently, only {CLK_TCK} is not required by POSIX.2.

This description does not specify the *name* values for the arguments to the various functions. This is because different language bindings might use different naming conventions, or might use a completely different scheme for obtaining the required configurable values. Specific names for the *name* values for the C language binding are given in B.10.2.

7.9 Locale Control

Any language binding to Language-Independent System Services shall include a facility to set locale control information.

The language-independent specification for this facility has not been developed. The C binding for this facility is described in B.11.

7.9.0.1 Locale Control Rationale. *(This subclause is not a part of P1003.2)*

This facility is required in POSIX.2 so that applications can control the locale, which affects the operation of POSIX.2 utilities.

Table 7-1 – POSIX.1 Numeric-Valued Configurable Variables

{ARG_MAX}	{NAME_MAX}	{_POSIX_CHOWN_RESTRICTED}
{CHILD_MAX}	{NGROUPS_MAX}	{_POSIX_JOB_CONTROL}
{LINK_MAX}	{OPEN_MAX}	{_POSIX_NO_TRUNC}
{MAX_CANON}	{PATH_MAX}	{_POSIX_SAVED_IDS}
{MAX_INPUT}	{PIPE_BUF}	{_POSIX_VDISABLE}

Annex A (normative)

C Language Development Utilities Option

1 This annex describes utilities used for the development of C language applica-
2 tions, including compilation or translation of C source code and complex program
3 generators for simple lexical tasks and processing of context-free grammars.

4 The utilities described in this annex may be provided by the conforming system;
5 however, any system claiming conformance to the **C Language Development**
6 **Utilities Option** shall provide all of the utilities described here. The utilities
7 described in Section 6 are prerequisites to this annex.

8 **A.0.1 C Language Development Utilities Option Rationale.** *(This subclause is* 9 *not a part of P1003.2)*

10 The portions of this standard that concern specific languages—currently C and
11 FORTRAN—have been collected to the rear of the document as Normative
12 Annexes. For purposes of conformance, they are no less a part of the standard
13 than one of the numbered sections. They were grouped as Annexes to illustrate
14 that the base standard is [planned to be] language independent, giving a small
15 degree of separation. The working group also wished to send a message to those
16 groups planning other language bindings: the standard is not C-oriented, and
17 there's plenty of room to add more annexes for your languages as you develop
18 them, right alongside C and FORTRAN.

A.1 `c89` — Compile Standard C programs

A.1.1 Synopsis

```
c89 [-c] [-D name[=value]] ... [-E] [-g] [-I directory] ... [-L directory] ...
    [-o outfile] [-O] [-s] [-U name] ... operand ...
```

A.1.2 Description

The `c89` utility is the interface to the standard C compilation system; it shall accept source code conforming to the C Standard {7}. The system conceptually consists of a compiler and link editor. The files referenced by *operands* shall be compiled and linked to produce an executable file. (It is unspecified whether the linking occurs entirely within the operation of `c89`; some systems may produce objects that are not fully resolved until the file is executed.)

If the `-c` option is specified, for all pathname operands of the form *file.c*, the files

```
$(basename pathname .c).o
```

shall be created as the result of successful compilation. If the `-c` option is not specified, it is unspecified whether such `.o` files are created or deleted for the *file.c* operands.

If there are no options that prevent link editing (such as `-c` or `-E`), and all operands compile and link without error, the resulting executable file shall be written according to the `-o outfile` option (if present) or to the file `a.out`.

The executable file shall be created as specified in 2.9.1.4, except that the file permissions shall be set to

```
S_IRWXO | S_IRWXG | S_IRWXU
```

(see 5.6.1.2 in POSIX.1 {8}) and that the bits specified by the *umask* of the process shall be cleared.

A.1.3 Options

The `c89` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that:

- The `-l library` operands have the format of options, but their position within a list of operands affects the order in which libraries are searched.
- The order of specifying the `-I` and `-L` options is significant.
- Conforming applications shall specify each option separately; that is, grouping option letters (e.g., `-cO`) need not be recognized by all implementations.

The following options shall be supported by the implementation:

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

53 -c Suppress the link-edit phase of the compilation, and do not
54 remove any object files that are produced.

55 -g Produce symbolic information in the object or executable files; the
56 nature of this information is unspecified, and may be modified by
57 implementation-defined interactions with other options.

58 -s Produce object and/or executable files from which symbolic and
59 other information not required for proper execution using *exec*
60 (see POSIX.1 {8} 3.1.2) has been removed (stripped). If both -g
61 and -s options are present, the action taken is unspecified.

62 -o *outfile* Use the pathname *outfile*, instead of the default *a.out*, for the
63 executable file produced. If the -o option is present with -c or
64 -E, the result is unspecified.

65 -D *name*[=*value*] Define *name* as if by a C-language #define directive. If no
66 =*value* is given, a value of 1 shall be used. The -D option has
67 lower precedence than the -U option. That is, if *name* is used in
68 both a -U and a -D option, *name* shall be undefined regardless of
69 the order of the options. Additional implementation-defined
70 names may be provided by the compiler. Implementations shall
71 support at least 2048 bytes of -D definitions and 256 *names*.
72

73 -E Copy C-language source files to the standard output, expanding
74 all preprocessor directives; no compilation shall be performed. If
75 any operand is not a text file, the effects are unspecified.

76 -I *directory* Change the algorithm for searching for headers whose names are
77 not absolute pathnames to look in the directory named by the
78 *directory* pathname before looking in the usual places. Thus,
79 headers whose names are enclosed in double-quotes (" ") shall be
80 searched for first in the directory of the file with the #include
81 line, then in directories named in -I options, and last in the
82 usual places. For headers whose names are enclosed in angle
83 brackets (<>), the header shall be searched for only in directories
84 named in -I options and then in the usual places. Directories
85 named in -I options shall be searched in the order specified.
86 Implementations shall support at least ten instances of this
87 option in a single c89 command invocation.
88

89 -L *directory* Change the algorithm of searching for the libraries named in the
90 -l objects to look in the directory named by the *directory* path-
91 name before looking in the usual places. Directories named in -L
92 options shall be searched in the order specified. Implementations
93 shall support at least ten instances of this option in a single c89
94 command invocation. If a directory specified by a -L option con-
95 tains files named *libc.a*, *libm.a*, *libl.a*, or *liby.a*, the
96 results are unspecified.
97

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

- 98 -O Optimize. The nature of the optimization is unspecified.
- 99 -U *name* Remove any initial definition of *name*.
- 100 Multiple instances of the -D, -I, -U, and -L options can be specified.

101 **A.1.4 Operands**

102 An *operand* is either in the form of a pathname or the form -l *library*. At least
 103 one operand of the pathname form shall be specified. The following operands
 104 shall be supported by the implementation:

- 105 *file.c* A C-language source file to be compiled and optionally linked.
 106 The operand shall be of this form if the -c option is used.
- 107 *file.a* A library of object files typically produced by ar (see 6.1), and
 108 passed directly to the link editor. Implementations may recog-
 109 nize implementation-defined suffixes other than .a as denoting
 110 object file libraries.
- 111 *file.o* An object file produced by c89 -c, and passed directly to the link
 112 editor. Implementations may recognize implementation-defined
 113 suffixes other than .o as denoting object files.

114 The processing of other files is implementation defined.

- 115 -l *library* (The letter ell.) Search the library named:

116 lib*library*.a

117 A library shall be searched when its name is encountered, so the
 118 placement of a -l operand is significant. Several standard
 119 libraries can be specified in this manner, as described in A.1.7.
 120 Implementations may recognize implementation-defined suffixes
 121 other than .a as denoting libraries.

122 **A.1.5 External Influences**

123 **A.1.5.1 Standard Input**

124 None.

125 **A.1.5.2 Input Files**

126 The input file shall be one of the following: a text file containing a C-language
 127 source program; an object file in the format produced by c89 -c; or a library of
 128 object files, in the format produced by archiving zero or more object files, using ar.
 129 Implementations may supply additional utilities that produce files in these for-
 130 mats. Additional input file formats are implementation defined.

131 **A.1.5.3 Environment Variables**

132 The following environment variables shall affect the execution of `c89`:

133	LANG	This variable shall determine the locale to use for the
134		locale categories when both LC_ALL and the correspond-
135		ing environment variable (beginning with LC_) do not
136		specify a locale. See 2.6.
137	LC_ALL	This variable shall determine the locale to be used to over-
138		ride any values for locale categories specified by the set-
139		tings of LANG or any environment variables beginning
140		with LC_ .
141	LC_CTYPE	This variable shall determine the locale for the interpreta-
142		tion of sequences of bytes of text data as characters (e.g.,
143		single- versus multibyte characters in arguments and
144		input files).
145	LC_MESSAGES	This variable shall determine the language in which mes-
146		sages should be written.
147	TMPDIR	This variable shall be interpreted as a pathname that
148		should override the default directory for temporary files, if
149		any.

150 **A.1.5.4 Asynchronous Events**

151 Default.

152 **A.1.6 External Effects**

153 **A.1.6.1 Standard Output**

154 If more than one file operand ending in `.c` (or possibly other unspecified suffixes)
155 is given, for each such file:

156 `"%s:\n", <file>`

157 may be written. These messages, if written, shall precede the processing of each
158 input file; they shall not be written to standard output if they are written to stan-
159 dard error, as described in A.1.6.2.

160 If the `-E` option is specified, the standard output shall be a text file that 1
161 represents the results of the preprocessing stage of the language; it may contain 1
162 extra information appropriate for subsequent compilation passes. 1

163 **A.1.6.2 Standard Error**

164 Used only for diagnostic messages. If more than one file operand ending in `.c` (or
165 possibly other unspecified suffixes) is given, for each such file:

166 `"%s:\n", <file>`

167 may be written to allow identification of the diagnostic and warning messages
168 with the appropriate input file. These messages, if written, shall precede the pro-
169 cessing of each input file; they shall not be written to the standard error if they
170 are written to the standard output, as described in A.1.6.1.

171 This utility may produce warning messages about certain conditions that do not
172 warrant returning an error (nonzero) exit value.

173 **A.1.6.3 Output Files**

174 Object files or executable files or both are produced in unspecified formats.

175 **A.1.7 Extended Description**

176 **A.1.7.1 Standard Libraries**

177 The `c89` utility shall recognize the following `-l` operands for standard libraries:

178 `-l c` This library contains all library functions referenced in `<stdlib.h>`,
179 `<stdio.h>`, `<time.h>`, `<setjmp.h>`, `<signal.h>`, `<unistd.h>`,
180 `<sys/types.h>`, `<string.h>`, and `<ctype.h>`, except for those
181 functions referenced in `<math.h>`. If an invocation of

182 `getconf _POSIX_VERSION`

183 exits with a status of zero, the library searched also shall include all
184 functions defined by POSIX.1 {8}; if the status is nonzero, it is
185 unspecified whether these functions are available. If an invocation of

186 `getconf _POSIX2_C_BIND`

187 exits with a status of zero, the library searched also shall include all
188 functions specified in Annex B; if the status is nonzero, it is
189 unspecified whether these functions are available. An implementa-
190 tion shall not require this operand to be present to cause a search of
191 this library.

192 `-l m` This library contains all functions referenced in `<math.h>`. An
193 implementation may search this library in the absence of this
194 operand.

195 `-l l` This library contains all functions required by the C-language output
196 of `lex` (see A.2) that are not made available through the `-l c`
197 operand.

198 `-l y` This library contains all functions required by the C-language output
 199 of `yacc` (see A.3) that are not made available through the `-l c`
 200 operand.

201 In the absence of options that inhibit invocation of the link editor, such as `-c` or
 202 `-E`, the `c89` utility shall cause the equivalent of a `-l c` operand to be passed to
 203 the link editor as the last `-l` operand, causing it to be searched after all other
 204 object files and libraries are loaded.

205 It is unspecified whether the libraries `libc.a`, `libm.a`, `libl.a`, and `liby.a`
 206 exist as regular files. The implementation may accept as `-l` operands names of
 207 objects that do not exist as regular files.

208 **A.1.7.2 External Symbols**

209 The C compiler and link editor shall support the significance of external symbols 1
 210 up to a length of at least 31 bytes; the action taken upon encountering symbols
 211 exceeding the implementation-defined maximum symbol length is unspecified.

212 The compiler and link editor shall support a minimum of 511 external symbols
 213 per source or object file, and a minimum of 4095 external symbols total. A diag-
 214 nostic message shall be written to the standard output if the implementation-
 215 defined limit is exceeded; other actions are unspecified.

216 **A.1.8 Exit Status**

217 The `c89` utility shall exit with one of the following values:

218 0 Successful compilation or link edit.
 219 >0 An error occurred.

220 **A.1.9 Consequences of Errors**

221 When `c89` encounters a compilation error that causes an object file not to be
 222 created, it shall write a diagnostic to standard error and continue to compile other
 223 source code operands, but it shall not perform the link phase and shall return a
 224 nonzero exit status. If the link edit is unsuccessful, a diagnostic message shall be
 225 written to standard error and `c89` shall exit with a nonzero status.

226 **A.1.10 Rationale.** *(This subclause is not a part of P1003.2)*

227 **Examples, Usage**

228 Note that some implementations support a finer-grained model of compilation
 229 than the one described above. In this model, the following conceptual phases may
 230 exist: preprocessor, compiler, optimizer, assembler, link editor. Such implemen-
 231 tations may support these additional options to the `c89` utility:

232 -P Preprocess, but do not compile, the named C programs and leave the
233 result on corresponding files suffixed `.i`.

234 -S Compile the named C programs into assembly language, and leave the
235 assembler-language output on corresponding files suffixed `.s`. No object
236 files are created.

237 [-Wc, *arg1* [, *arg2* ...]]

238 Hand off the argument(s) *argi* to phase *c* where *c* is one of [p02a1] indi-
239 cating preprocessor, compiler, optimizer, assembler, or link editor,
240 respectively. For example, -Wa, -m passes -m to the assembler phase.
241 (Note the rationale concerning -W in 2.10.1.1.)

242 The -fpq options have been excluded, since they use features that are not in this
243 standard.

244 In specifying that *file.a* operands are *typically* produced by `ar`, it is the intention
245 of POSIX.2 to require that object libraries produced by `ar` be usable by `c89`, but
246 not to preclude an implementation from supplying another utility that creates
247 object library files.

248 The following are examples of usage:

249 c89 -o foo foo.c Compiles `foo.c` and creates the executable `foo`.

250 c89 -c foo.c Compiles `foo.c` and creates the object file `foo.o`.

251 c89 foo.c Compiles `foo.c` and creates the executable `a.out`.

252 c89 foo.c bar.o Compiles `foo.c`, links it with `bar.o`, and creates the
253 executable `a.out`. Also creates and leaves `foo.o`.

254 The following examples clarify the use and interactions of -L options and -l
255 operands:

256 Consider the case in which module `a.c` calls function `f()` in library `libQ.a`,
257 and module `b.c` calls function `g()` in library `libp.a`. Assume that both
258 libraries reside in `/a/b/c`. The command line to compile and link in the
259 desired way is:

260 c89 -L /a/b/c main.o a.c -l Q b.c -l p

261 In this case the `-l Q` operand need only precede the first `-l p` operand,
262 since both `libQ.a` and `libp.a` reside in the same directory.

263 Multiple -L operands can be used when library name collisions occur.
264 Building on the previous example, suppose that we now want to use a new
265 `libp.a`, in `/a/a/a`, but we still want `f()` from `/a/b/c/libQ.a`.

266 c89 -L /a/a/a -L /a/b/c main.o a.c -l Q b.c -l p

267 In this example, the linker searches the -L options in the order specified,
268 and finds `/a/a/a/libp.a` before `/a/b/c/libp.a` when resolving refer-
269 ences for `b.c`. The order of the -l operands is still important, however.

270 There is the possible implication that if a user supplies versions of the standard
271 library functions (before they would be encountered by an implicit `-l c` or explicit

`-l m`), that those versions would be used in place of the standard versions. There are various reasons this might not be true (functions defined as macros, manipulations for clean namespace, etc.), so the existence of files named in the same manner as the standard libraries within the `-L` directories is explicitly stated to produce unspecified behavior.

Some historical implementations have permitted `-L` options to be interspersed with `-l` operands on the command line; with respect to POSIX, such behavior would be considered a vendor extension. For an application to compile consistently on systems that do not behave like this, it is necessary for a conforming application to supply all `-L` options before any of the `-l` options.

Some historical implementations have created `.o` files when `-c` is not specified and more than one source file is given. Since this area is left unspecified, the application cannot rely on `.o` files being created, but it also must be prepared for any related `.o` files that already exist being deleted at the completion of the link edit.

History of Decisions Made

The name of this utility differs from the historical `cc` name. The C Standard {7} document was approved during the development of POSIX.2, and it is clear that POSIX must support Standard C; there is no other good way of specifying a C language. The support of the C Standard {7} by `c89` also mandates the Standard C math libraries. An alternative approach was considered: provide an option to select the type of compilation required. However, it was found that all available option letters were already in use in the various historical `cc` utilities. Thus, this name change is being used essentially as a switch. There was some temptation to use the name change as an excuse to mandate a cleaner interface (e.g., conform to the utility syntax guidelines), but this was resisted; the majority of early `c89` implementations are expected to be satisfied with historical `ccs` with only minimal changes. This was decided more from the standpoint of existing applications and makefiles than for the implementors' sake.

The `-l library` operand must be capable of being interspersed with file name operands so that the order in which libraries are searched by the link editor can be specified.

The search algorithm for `-I directory` states that the directory of the file with the `#include` file is searched first, rather than being implementation defined. It is believed that this reflects most implementations, and it disallows variations on different implementations, since this would make it very difficult to distribute source code in a compatible form.

The `-I` options are searched in the order specified (which is left to right in English). This resolves the conflict of what header file is used if multiple files with the same name exist in different directories in the `include` path.

In a future extension or supplement to this standard, *should* will be changed to *shall* with respect to support for **TMPDIR** by applications.

It is unclear whether `c89` requires such a large number of file descriptors that its requirement should be documented here; POSIX.2 remains silent on the issue. It is also noted that an undocumented feature of some C compilers is that if file descriptor 9 is open, a linkage trace is written to it.

There is no pseudo-*printf()* specification for compile errors because no common format could be identified. As new C compilers are written, they are encouraged to use the following format:

```
"%s: %s: %d %s\n", <compiler phase>, <file name>, <line number>,
<explanation>
```

The following option proposals were considered and rejected:

- (1) The `-M` option in BSD does not exist in System V, and is not seen to enhance application portability.
- (2) The `-S` option was not seen to enhance application portability, and makes assumptions about the underlying architecture.

Earlier drafts included a `-v` option to select a compiler version. Not only did this letter (and every other upper- and lowercase letter) collide with one historical implementation or another, but there was no agreement on how many compiler versions should be defined, or what they should mean. Another choice is to specify that the `cc` utility invoke a Standard C compiler. By specifying `c89` instead, an installation is able to link either a “common usage” or a Standard C compiler to the name `cc`. Implementors are free to select implementation-defined options to select (nonportable) extensions to their existing C compiler to aid the transition to Standard C.

The `-g` and `-s` options are not specified as mutually exclusive. Historically these two options have been mutually exclusive, but because both are so loosely specified, it seemed cleaner to leave their interaction unspecified.

The `-E` option was added because headers are not required to be separate files in a POSIX.1-conformant system; these values could be hard-coded into the compiler, or might only be accessible in a nonportable way. Hence, while not strictly required for application portability, this option is a practical necessity as a portable means for ascertaining the real effects of preprocessor statements.

In BSD systems, using `-c` and `-o` in the same command causes the object module to be stored in the specified file. In System V, this produces an error condition. Therefore, POSIX.2 indicates that this is an unspecified condition.

Reasonably precise specification of standard library access is required. Implementations are not required to have `/usr/lib/libc.a`, etc., as many historical implementations do, but if not they are required to recognize `c`, `m`, `l`, and `y` as tokens. Libraries `l` and `y` can be empty if the library functions specified for `lex` and `yacc` are accessible through the `-l c` operand. Historically, these libraries have been necessary, but they are not required for a conforming implementation.

External symbol size limits are in a normative subclause; portable applications need to know these limits. However, the minimum maximum symbol length should be taken as a constraint on a portable application, not on an

implementation, and consequently the action taken for a symbol exceeding the limit is unspecified. The minimum size for the external symbol table was added for similar reasons.

The Consequences of Errors subclause clearly specifies the compiler's behavior when compilation or link-edit error occur. The behavior of several historical implementations was examined, and the choice was made to be silent on the status of the executable, or `a.out`, file in the face of compiler or linker errors. If a linker writes the executable file, then links it on disk with `lseek()`s and `write()`s, the partially-linked executable can be left on disk and its execute bits turned off if the link edit fails. However, if the linker links the image in memory before writing the file to disk, it need not touch the executable file (if it already exists) because the link edit fails. Since both approaches are existing practice, a portable application shall rely on the exit status of `c89`, rather than on the existence or mode of the executable file.

The requirement that portable applications specify compiler options separately is to reserve the multicharacter option namespace for vendor-specific compiler options, which are known to exist in many historical implementations. Implementations are not required to recognize, for example `-gc` as if it were `-g -c`; nor are they forbidden from doing so. The synopsis shows all of the options separately to highlight this requirement on applications.

Echoing filenames to standard error is considered a diagnostic message, because it might otherwise be difficult to associate an error message with the erring file. The text specifies either standard error or standard output for these messages because some historical practice uses standard output, but there was considerable sentiment expressed for allowing it to be on standard error instead. The rationale for using standard output is that these are not really error message headers, but a running progress report on which files have been processed. The messages are described as optional because there might be different ways of constructing the compiler's messages that should not be precluded.

A.2 lex — Generate programs for lexical tasks

A.2.1 Synopsis

`lex [-t] [-n | -v] [file ...]`

Obsolescent Version:

`lex -c [-t] [-n | -v] [file ...]`

A.2.2 Description

The `lex` utility shall generate C programs to be used in lexical processing of character input, and that can be used as an interface to `yacc` (see A.3). The C programs shall be generated from `lex` source code and conform to the C Standard {7}. Usually, the `lex` utility writes the program it generates to the file `lex.yy.c`; the state of this file is unspecified if `lex` exits with a nonzero exit status. See A.2.7 for a complete description of the `lex` input language.

A.2.3 Options

The `lex` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

- `-c` (Obsolescent.) Indicate C-language action (default option).
- `-n` Suppress the summary of statistics usually written with the `-v` option. If no table sizes are specified in the `lex` source code and the `-v` option is not specified, then `-n` is implied.
- `-t` Write the resulting program to standard output instead of `lex.yy.c`.
- `-v` Write a summary of `lex` statistics to the standard output. (See the discussion of `lex` table sizes in A.2.7.1.) If the `-t` option is specified and `-n` is not specified, this report shall be written to standard error. If table sizes are specified in the `lex` source code, and if the `-n` option is not specified, the `-v` option may be enabled.

A.2.4 Operands

The following operand shall be supported by the implementation:

416 *file* A pathname of an input file. If more than one such *file* is
 417 specified, all files shall be concatenated to produce a single `lex`
 418 program. If no *file* operands are specified, or if a *file* operand is `-`,
 419 the standard input shall be used.

420 **A.2.5 External Influences**

421 **A.2.5.1 Standard Input**

422 The standard input shall be used if no *file* operands are specified, or if a *file*
 423 operand is `-`. See Input Files.

424 **A.2.5.2 Input Files**

425 The input files shall be text files containing `lex` source code, as described in
 426 A.2.7.

427 **A.2.5.3 Environment Variables**

428 The following environment variables shall affect the execution of `lex`:

429 LANG	This variable shall determine the locale to use for the
430	locale categories when both LC_ALL and the correspond-
431	ing environment variable (beginning with LC_) do not
432	specify a locale. See 2.6.
433 LC_ALL	This variable shall determine the locale to be used to over-
434	ride any values for locale categories specified by the set-
435	tings of LANG or any environment variables beginning
436	with LC_ .
437 LC_COLLATE	This variable shall determine the locale for the behavior of
438	ranges, equivalence classes, and multicharacter collating
439	elements within regular expressions. If this variable is
440	not set to the POSIX Locale, the results are unspecified.
441 LC_CTYPE	This variable shall determine the locale for the interpreta-
442	tion of sequences of bytes of text data as characters (e.g.,
443	single- versus multibyte characters in arguments and
444	input files) and the behavior of character classes within
445	extended regular expressions. If this variable is not set to
446	the POSIX Locale, the results are unspecified.
447 LC_MESSAGES	This variable shall determine the language in which mes-
448	sages should be written.

A.2.5.4 Asynchronous Events

Default.

A.2.6 External Effects

A.2.6.1 Standard Output

If the `-t` option is specified, the text file of C source code output of `lex` shall be written to standard output.

If the `-t` option is not specified:

- (1) Implementation-defined informational, error, and warning messages concerning the contents of `lex` source code input shall be written to either the standard output or standard error.
- (2) If the `-v` option is specified and the `-n` option is not specified, `lex` statistics shall also be written to either the standard output or standard error, in an implementation-defined format. These statistics may also be generated if table sizes are specified with a `%` operator in the *Definitions* section (see A.2.7), as long as the `-n` option is not specified.

A.2.6.2 Standard Error

If the `-t` option is specified, implementation-defined informational, error, and warning messages concerning the contents of `lex` source code input shall be written to the standard error.

If the `-t` option is not specified:

- (1) Implementation-defined informational, error, and warning messages concerning the contents of `lex` source code input shall be written to either the standard output or standard error.
- (2) If the `-v` option is specified and the `-n` option is not specified, `lex` statistics shall also be written to either the standard output or standard error, in an implementation-defined format. These statistics may also be generated if table sizes are specified with a `%` operator in the *Definitions* section (see A.2.7), as long as the `-n` option is not specified.

A.2.6.3 Output Files

A text file containing C source code shall be written to `lex.yy.c`, or to the standard output if the `-t` option is present.

A.2.7 Extended Description

Each input file contains `lex` source code, which is a table of regular expressions with corresponding actions in the form of C program fragments.

When `lex.yy.c` is compiled and linked with the `lex` library (using the `-l 1` operand with `c89`), the resulting program reads character input from the standard input and partitions it into strings that match the given expressions.

When an expression is matched, these actions shall occur:

- The input string that was matched is left in *yytext* as a null-terminated string; *yytext* is either an external character array or a pointer to a character string. As explained in A.2.7.1, the type can be explicitly selected using the `%array` or `%pointer` declarations, but the default is implementation defined.
- The external *int yyleng* is set to the length of the matching string.
- The expression's corresponding program fragment, or action, is executed.

During pattern matching, `lex` shall search the set of patterns for the single longest possible match. Among rules that match the same number of characters, the rule given first shall be chosen.

The general format of `lex` source is:

Definitions

`%%`

Rules

`%%`

User Subroutines

The first `%%` is required to mark the beginning of the rules (regular expressions and actions); the second `%%` is required only if user subroutines follow.

Any line in the *Definitions* section beginning with a `<blank>` shall be assumed to be a C program fragment and shall be copied to the external definition area of the `lex.yy.c` file. Similarly, anything in the *Definitions* section included between delimiter lines containing only `%{` and `%}` shall also be copied unchanged to the external definition area of the `lex.yy.c` file.

Any such input (beginning with a `<blank>` or within `%{` and `%}` delimiter lines) appearing at the beginning of the *Rules* section before any rules are specified shall be written to `lex.yy.c` after the declarations of variables for the `yylex()` function and before the first line of code in `yylex()`. Thus, user variables local to `yylex()` can be declared here, as well as application code to execute upon entry to `yylex()`.

The action taken by `lex` when encountering any input beginning with a `<blank>` or within `%{` and `%}` delimiter lines appearing in the *Rules* section but coming after one or more rules is undefined. The presence of such input may result in an erroneous definition of the `yylex()` function.

A.2.7.1 `lex` Definitions

Definitions appear before the first `%%` delimiter. Any line in this section not contained between `%{` and `%}` lines and not beginning with a `<blank>` shall be assumed to define a `lex` substitution string. The format of these lines is:

name substitute

If a *name* does not meet the requirements for identifiers in the C Standard [7], the result is undefined. The string *substitute* shall replace the string *{name}* when it is used in a rule. The *name* string shall be recognized in this context only when the braces are provided and when it does not appear within a bracket expression or within double-quotes.

In the *Definitions* section, any line beginning with a `%` (percent-sign) character and followed by an alphanumeric word beginning with either `s` or `S` shall define a set of start conditions. Any line beginning with a `%` followed by a word beginning with either `x` or `X` shall define a set of exclusive start conditions. When the generated scanner is in a `%s` state, patterns with no state specified shall be also active; in a `%x` state, such patterns shall not be active. The rest of the line, after the first word, shall be considered to be one or more `<blank>`-separated names of start conditions. Start condition names shall be constructed in the same way as definition names. Start conditions can be used to restrict the matching of regular expressions to one or more states as described in the section A.2.7.4.

Implementations shall accept either of the following two mutually exclusive declarations in the *Definitions* section:

`%array` Declare the type of *yytext* to be a null-terminated character array.

`%pointer` Declare the type of *yytext* to be a pointer to a null-terminated character string.

The default type of *yytext* is implementation defined. If an application refers to *yytext* outside of the scanner source file (i.e., via an `extern`), the application shall include the appropriate `%array` or `%pointer` declaration in the scanner source file.

Implementations shall accept declarations in the *Definitions* section for setting certain internal table sizes. The declarations are shown in Table A-1. In the table, *n* represents a positive decimal integer, preceded by one or more `<blank>`s. The exact meaning of these table size numbers is implementation defined. The implementation shall document how these numbers affect the `lex` utility and how they are related to any output that may be generated by the implementation should space limitations be encountered during the execution of `lex`. It shall be possible to determine from this output which of the table size values needs to be modified to permit `lex` to successfully generate tables for the input language. The values in the column Minimum Value represent the lowest values conforming implementations shall provide.

Table A-1 – lex Table Size Declarations

Declaration	Description	Minimum Value
%p <i>n</i>	Number of positions	2500
%n <i>n</i>	Number of states	500
%a <i>n</i>	Number of transitions	2000
%e <i>n</i>	Number of parse tree nodes	1000
%k <i>n</i>	Number of packed character classes	1000
%o <i>n</i>	Size of the output array	3000

A.2.7.2 lex Rules

The rules in lex source files are a table in which the left column contains regular expressions and the right column contains actions (C program fragments) to be executed when the expressions are recognized.

ERE action

ERE action

...

The extended regular expression (*ERE*) portion of a rule shall be separated from *action* by one or more <blank>s. A regular expression containing <blank>s shall be recognized under the following conditions: the entire expression appears within double-quotes; or, the <blank>s appear within double-quotes or square brackets; or, each <blank> is preceded by a backslash character.

A.2.7.3 lex User Subroutines

Anything in the user subroutines section shall be copied to lex.yy.c following `yylex()`.

A.2.7.4 lex Regular Expressions

The lex utility shall support the set of extended regular expressions (see 2.8.4), with the following additions and exceptions to the syntax:

" . . . " Any string enclosed in double-quotes shall represent the characters within the double-quotes as themselves, except that backslash escapes (which appear in Table A-2) shall be recognized. Any backslash-escape sequence shall be terminated by the closing quote. For example, "\01" "1" represents a single string: the octal value 1 followed by the character 1.

<state>*r*

<state1, state2, ...>*r*

The regular expression *r* shall be matched only when the program

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

598		is in one of the start conditions indicated by <i>state</i> , <i>state1</i> , etc.; see	1
599		A.2.7.5. (As an exception to the typographical conventions of the	
600		rest of this standard, in this case <i><state></i> does not represent a	
601		metavariable, but the literal angle-bracket characters surround-	
602		ing a symbol.) The start condition shall be recognized as such	1
603		only at the beginning of a regular expression.	1
604	<i>r/x</i>	The regular expression <i>r</i> shall be matched only if it is followed by	
605		an occurrence of regular expression <i>x</i> . The token returned in	
606		<i>yytext</i> shall only match <i>r</i> . If the trailing portion of <i>r</i> matches the	
607		beginning of <i>x</i> , the result is unspecified. The <i>r</i> expression cannot	
608		include further trailing context or the \$ (match-end-of-line) opera-	
609		tor; <i>x</i> cannot include the ^ (match-beginning-of-line) operator, nor	
610		trailing context, nor the \$ operator. That is, only one occurrence	
611		of trailing context is allowed in a <i>lex</i> regular expression, and the	
612		^ operator only can be used at the beginning of such an expres-	
613		sion.	
614	{ <i>name</i> }	When <i>name</i> is one of the substitution symbols from the	
615		<i>Definitions</i> section (see A.2.7.1), the string, including the enclos-	
616		ing braces, shall be replaced by the <i>substitute</i> value. The <i>substi-</i>	
617		<i>tute</i> value shall be treated in the extended regular expression as	
618		if it were enclosed in parentheses. No substitution shall occur if	
619		{ <i>name</i> } occurs within a bracket expression or within double-	
620		quotes.	
621	Within an ERE, a backslash character shall be considered to begin an escape		
622	sequence as specified in Table 2-15 (see 2.12). In addition, the escape sequences		
623	in Table A-2 shall be recognized.		
624	A literal <i><newline></i> character cannot occur within an ERE; the escape sequence		1
625	\n can be used to represent a <i><newline></i> . A <i><newline></i> shall not be matched by		2
626	a period operator.		2
627	The order of precedence given to extended regular expressions for <i>lex</i> differs from		2
628	that specified in Table 2-13. The order of precedence for <i>lex</i> shall be as shown in		
629	Table A-3, from high to low.		
630	NOTE: The escaped characters entry is not meant to imply that these are operators, but they are		2
631	included in the table to show their relationships to the true operators. The start condition, trailing		2
632	context, and anchoring notations have been omitted from the table because of the placement restric-		2
633	tions described in this subclause; they can only appear at the beginning or ending of an ERE.		2
634	The ERE anchoring operators (^ and \$) do not appear in Table A-3. With <i>lex</i> reg-		2
635	ular expressions, these operators are restricted in their use: the ^ operator can		2
636	only be used at the beginning of an entire regular expression, and the \$ operator		2
637	only at the end. The operators apply to the entire regular expression. Thus, for		2
638	example, the pattern (^abc) (def\$) is undefined; it can instead be written as		2
639	two separate rules, one with the regular expression ^abc and one with def\$,		2
640	which share a common action via the special action (see below). If the pattern		2
641	were written ^abc def\$, it would match either of abc or def on a line by itself.		2
642	Note also that \$ is a form of trailing context (it is equivalent to /\n) and as such		2

Table A-2 – lex Escape Sequences

Escape Sequence	Description	Meaning	
<code>\digits</code>	<backslash> followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0, (i.e., representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the one-, two-, or three-digit octal integer. If the size of a byte on the system is greater than nine bits, the valid escape sequence used to represent a byte is implementation-defined. Multibyte characters require multiple, concatenated escape sequences of this type, including the leading <code>\</code> for each byte.	1 1 1 1 1 1 1 1 1 1
<code>\xdigits</code>	<backslash> followed by the longest sequence of hexadecimal-digit characters (01234567abcdefABCDEF). If all of the digits are 0, (i.e., representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the hexadecimal integer.	1 1 1 1 1 1
<code>\c</code>	<backslash> followed by any character not described in this table or in Table 2-15	The character <i>c</i> , unchanged.	

Table A-3 – lex ERE Precedence

<i>collation-related bracket symbols</i>	[= =] [: :] [. .]	
<i>escaped characters</i>	\<special character>	1
<i>bracket expression</i>	[]	1
<i>quoting</i>	" . . . "	1
<i>grouping</i>	()	1
<i>definition</i>	{ name }	1
<i>single-character RE duplication</i>	* + ?	1
<i>concatenation</i>		1
<i>interval expression</i>	{ m , n }	2
<i>alternation</i>		2

cannot be used with regular expressions containing another instance of the operator (see the preceding discussion of trailing context). 2

The additional regular expressions trailing-context operator `/` can be used as an ordinary character if presented within double-quotes, `" / "`; preceded by a backslash, `\ /`; or within a bracket expression, `[/]`. The start-condition `<` and `>` operators shall be special only in a start condition at the beginning of a regular 1

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

expression; elsewhere in the regular expression they shall be treated as ordinary characters. 1
1

688 A.2.7.5 `lex` Actions

689 The action to be taken when an *ERE* is matched can be a C program fragment or
690 the special actions described below; the program fragment can contain one or
691 more C statements, and can also include special actions. The empty C statement
692 `;` shall be a valid action; any string in the `lex.yy.c` input that matches the pat-
693 tern portion of such a rule is effectively ignored or skipped. However, the absence
694 of an action shall not be valid, and the action `lex` takes in such a condition is
695 undefined.

696 The specification for an action, including C statements and/or special actions, can
697 extend across several lines if enclosed in braces:

```
698     ERE <blank(s)> { program statement
699                     program statement }
```

700 The default action when a string in the input to a `lex.yy.c` program is not
701 matched by any expression shall be to copy the string to the output. Because the
702 default behavior of a program generated by `lex` is to read the input and copy it to
703 the output, a minimal `lex` source program that has just `%%` shall generate a C
704 program that simply copies the input to the output unchanged.

705 Four special actions shall be available: `|`, `ECHO;`, `REJECT;`, and `BEGIN`: 1

706 | The action `|` means that the action for the next rule is the action
707 for this rule. Unlike the other three actions, `|` cannot be enclosed
708 in braces or be semicolon-terminated; it shall be specified alone,
709 with no other actions.

710 `ECHO;` Write the contents of the string *yytext* on the output. 1

711 `REJECT;` Usually only a single expression is matched by a given string in 1
712 the input. `REJECT` means “continue to the next expression that
713 matches the current input,” and causes whatever rule was the
714 second choice after the current rule to be executed for the same
715 input. Thus, multiple rules can be matched and executed for one
716 input string or overlapping input strings. For example, given the
717 regular expressions `xyz` and `xy` and the input `xyz`, usually only
718 the regular expression `xyz` would match. The next attempted
719 match would start after `z`. If the last action in the `xyz` rule is
720 `REJECT`, both this rule and the `xy` rule would be executed. The
721 `REJECT` action may be implemented in such a fashion that flow of
722 control does not continue after it, as if it were equivalent to a
723 `goto` to another part of `yylex()`. The use of `REJECT` may result in
724 somewhat larger and slower scanners.

725 `BEGIN` The

```
726     BEGIN newstate;
```

727 action switches the state (start condition) to *newstate*. If the
 728 string *newstate* has not been declared previously as a start condi-
 729 tion in the *Definitions* section, the results are unspecified. The
 730 initial state is indicated by the digit 0 or the token INITIAL.

731 The functions or macros described below are accessible to user code included in
 732 the `lex` input. It is unspecified whether they appear in the C code output of `lex`,
 733 or are accessible only through the `-l 1` operand to `c89` (the `lex` library).

734 `int yylex(void)` Performs lexical analysis on the input; this is the pri-
 735 mary function generated by the `lex` utility. The func-
 736 tion shall return zero when the end of input is reached;
 737 otherwise it shall return nonzero values (tokens) deter-
 738 mined by the actions that are selected.

739 `int yymore(void)` When called, indicates that when the next input string
 740 is recognized, it is to be appended to the current value
 741 of *yytext* rather than replacing it; the value in *yyleng*
 742 shall be adjusted accordingly.

743 `int yyless(int n)` Retains *n* initial characters in *yytext*, NUL-terminated,
 744 and treats the remaining characters as if they had not
 745 been read; the value in *yyleng* shall be adjusted accord-
 746 ingly.

747 `int input(void)` Returns the next character from the input, or zero on
 748 end of file. It shall obtain input from the stream
 749 pointer *yyin*, although possibly via an intermediate
 750 buffer. Thus, once scanning has begun, the effect of
 751 altering the value of *yyin* is undefined. The character
 752 read is removed from the input stream of the scanner
 753 without any processing by the scanner.

754 `int unput(int c)` Returns the character *c* to the input; *yytext* and *yyleng*
 755 are undefined until the next expression is matched.
 756 The result of *unputting* more characters than have
 757 been input is unspecified.

758 The following functions appear only in the `lex` library accessible through the
 759 `-l 1` operand; they can therefore be redefined by a portable application:

760 `int yywrap(void)` Called by *yylex()* at end of file; the default *yywrap()*
 761 always shall return 1. If the application requires
 762 *yylex()* to continue processing with another source of
 763 input, then the application can include a function
 764 *yywrap()*, which associates another file with the exter-
 765 nal variable `FILE *yyin` and shall return a value of
 766 zero.

767 `int main(int argc, char *argv[])`
 768 Calls *yylex()* to perform lexical analysis, then exits.
 769 The user code can contain *main()* to perform
 770 application-specific operations, calling *yylex()* as

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

applicable.

Except for *input()*, *unput()*, and *main()*, all external and static names generated by *lex* shall begin with the prefix *yy* or *YY*.

A.2.8 Exit Status

The *lex* utility shall exit with one of the following values:

- 0 Successful completion.
- >0 An error occurred.

A.2.9 Consequences of Errors

Default.

A.2.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The following is an example of a *lex* program that implements a rudimentary scanner for a Pascal-like syntax:

```
%{
785 /* need this for the call to atof() below */
786 #include <math.h>
787 /* need this for printf(), fopen(), and stdin below */
788 #include <stdio.h>
789 %}

790 DIGIT      [0-9]
791 ID         [a-z][a-z0-9]*
792 %%

793 {DIGIT}+   {
794             printf("An integer: %s (%d)\n", yytext,
795                 atoi(yytext));
796             }

797 {DIGIT}+"."{DIGIT}*   {
798             printf("A float: %s (%g)\n", yytext,
799                 atof(yytext));
800             }

801 if|then|begin|end|procedure|function      {
802             printf("A keyword: %s\n", yytext);
803             }

804 {ID}       printf("An identifier: %s\n", yytext);

805 "+"|"-"|"*"|"|"          printf("An operator: %s\n", yytext);

806 "{"[^}\n]*"              /* eat up one-line comments */
```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```

807 [ \t\n]+          /* eat up white space */
808 .      printf("Unrecognized character: %s\n", yytext);
809 %%
810 int main(int argc, char *argv[ ])
811 {
812     ++argv, --argc; /* skip over program name */
813     if (argc > 0)
814         yyin = fopen(argv[0], "r");
815     else
816         yyin = stdin;
817     yylex();
818 }

```

819 The following examples have been included to clarify the differences between `lex`
820 regular expressions and regular expressions appearing elsewhere in this docu-
821 ment. For regular expressions of the form r/x , the string matching r is always
822 returned; confusion may arise when the beginning of x matches the trailing por-
823 tion of r . For example, given the regular expression $a*b/cc$ and the input
824 `aaabcc`, `yytext` would contain the string `aaab` on this match. But given the regu-
825 lar expression $x*/xy$ and the input `xxxxy`, the token `xxx`, not `xx`, is returned by
826 some implementations because `xxx` matches $x*$.

827 In the rule $ab*/bc$, the $b*$ at the end of r will extend r 's match into the beginning
828 of the trailing context, so the result is unspecified. If this rule were ab/bc , how-
829 ever, the rule matches the text `ab` when it is followed by the text `bc`. In this
830 latter case, the matching of r cannot extend into the beginning of x , so the result
831 is specified.

832 Unlike the general ERE rules, embedded anchoring is not allowed by most histori- 2
833 cal `lex` implementations. An example of embedded anchoring would be for pat- 2
834 terns such as $(^|)foo(|\$)$ to match `foo` when it exists as a complete word. 2
835 This functionality can be obtained using existing `lex` features: 2

```

836     ^foo/[ \n]      | 2
837     " foo"/[ \n]    /* found foo as a separate word */ 2

```

838 The precedence of regular expressions in `lex` does not match that of extended reg-
839 ular expressions in Section 2 because of historical practice. In System V `lex` and
840 its predecessors, a regular expression of the form $ab\{3\}$ matches `ababab`; an
841 ERE, such as used by `egrep`, would match `abbb`. Changing this precedence for
842 uniformity with `egrep` would have been desirable, but too many applications
843 would break in nonobvious ways.

844 Conforming applications are warned that in the *Rules* section, an *ERE* without an
845 action is not acceptable, but need not be detected as erroneous by `lex`. This may
846 result in compilation or run-time errors.

847 The purpose of `input()` is to take characters off the input stream and discard them
848 as far as the lexical analysis is concerned. A common use is to discard the body of
849 a comment once the beginning of a comment is recognized.

History of Decisions Made

Even though the `-c` option and references to the C language are retained in this description, `lex` may be generalized to other languages, as was done at one time for EFL, Extended FORTRAN Language. Since the `lex` input specification is essentially language independent, versions of this utility could be written to produce Ada, Modula-2, or Pascal code, and there are known historical implementations that do so.

The current description of `lex` bypasses the issue of dealing with internationalized regular expressions in the `lex` source code or generated lexical analyzer. If it follows the model used by `awk`, (the source code is assumed to be presented in the POSIX Locale, but input and output are in the locale specified by the environment variables), then the tables in the lexical analyzer produced by `lex` would interpret regular expressions specified in the `lex` source in terms of the environment variables specified when `lex` was executed. The desired effect would be to have the lexical analyzer interpret the regular expressions given in the `lex` source according to the environment specified when the lexical analyzer is executed, but this is not possible with the current `lex` technology.

Major international vendors believe that only limited internationalization is required for the POSIX.2 `lex`. The theoretically desirable goal of runtime-selectable locales is not feasible in the near future. Furthermore, the very nature of the lexical analyzers produced by `lex` must be closely tied to the lexical requirements of the input language being described, which will frequently be locale-specific anyway. (For example, writing an analyzer that is used for French text will not automatically be useful for processing other languages.) The text in the Environment Variable subclause allows locale-specific regular expression handling, but mandates only something similar to that provided in historical implementations.

The description of octal- and hexadecimal-digit escape sequences agrees with the C Standard {7} use of escape sequences. See the rationale for `ed` for a discussion of bytes larger than nine bits being represented by octal values. Hexadecimal values can represent larger bytes and multibyte characters directly, using as many digits as required.

There is no detailed output format specification. The observed behavior of `lex` under four different historical implementations was that none of these implementations consistently reported the line numbers for error and warning messages. Furthermore, there was a desire that `lex` be allowed to output additional diagnostic messages. Leaving message formats unspecified sidesteps these formatting questions and also avoids problems with internationalization.

Although the `%x` specifier for exclusive start conditions is not existing practice, it is believed to be a minor change to historical implementations, and greatly enhances the usability of `lex` programs since it permits an application to obtain the expected functionality with fewer statements.

The `%array` and `%pointer` declarations were added as a compromise between historical systems. The System V-based `lex` has copied the matched text to a

yytext array. The flex program, supported in BSD and GNU systems, uses a pointer. In the latter case, significant performance improvements are available for some scanners. Most existing programs should require no change in porting from one system to another because the string being referenced is null-terminated in both cases. (The method used by flex in its case is to null-terminate the token in-place by remembering the character that used to come right after the token and replacing it before continuing on to the next scan.) Multifile programs with external references to yytext outside the scanner source file should continue to operate on their existing systems, but would require one of the new declarations to be considered strictly portable.

The description of regular expressions avoids unnecessary duplication of regular expression details. Specifically, the | operator and {m,n} interval expression are not listed in A.2.7.4 because their meanings within a lex regular expression are the same as that for extended regular expressions.

The reason for the undefined condition associated with text beginning with a <blank> or within %{ and %} delimiter lines appearing in the Rules section is historical practice. Both BSD and System V lex copy the indented (or enclosed) input in the Rules section (except at the beginning) to unreachable areas of the yylex() function (the code is written directly after a break statement). In some cases, the System V lex generates an error message or a syntax error, depending on the form of indented input.

The intention in breaking the list of functions into those that may appear in lex.yy.c versus those that only appear in libl.a is that only those functions in libl.a can be reliably redefined by a portable application.

The descriptions of Standard Output and Standard Error are somewhat complicated because historical lex implementations chose to issue diagnostic messages to standard output (unless -t was given). POSIX.2 allows this behavior, but leaves an opening for the more expected behavior of using standard error for diagnostics. Also, the System V behavior of writing the statistics when any table sizes are given is allowed, while BSD-derived systems can avoid it. The programmer can always precisely obtain the desired results by using either the -t or -n options.

The Operands subclause does not mention the use of - as a synonym for standard input; not all historical implementations support such usage for any of the file operands.

The description of the Translation Table was deleted from earlier drafts because of its relatively low usage in historical applications.

The change to the definition of the input() function that allows buffering of input presents the opportunity for major performance gains in some applications.

A.3 yacc — Yet another compiler compiler

A.3.1 Synopsis

yacc [-dltv] [-b *file_prefix*] [-p *sym_prefix*] *grammar*

A.3.2 Description

The `yacc` utility shall read a description of a context-free grammar in *file* and write C source code, conforming to the C Standard [7], to a code file, and optionally header information into a header file, in the current directory. The C code shall define a function and related routines and macros for an automaton that executes a parsing algorithm meeting the requirements in A.3.7.8.

The form and meaning of the grammar is described in A.3.7.

The C source code and header file shall be produced in a form suitable as input for the C compiler (see `c89` in A.1).

A.3.3 Options

The `yacc` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

 -b *file_prefix*

 Use *file_prefix* instead of `y` as the prefix for all output filenames. The code file `y.tab.c`, the header file `y.tab.h` (created when `-d` is specified), and the description file `y.output` (created when `-v` is specified), shall be changed to *file_prefix.tab.c*, *file_prefix.tab.h*, and *file_prefix.output*, respectively.

 -d

 Write the header file; by default only the code file is written.

 -l

 Produce a code file that does not contain any `#line` constructs. If this option is not present, it is unspecified whether the code file or header file contains `#line` directives.

 -p *sym_prefix*

 Use *sym_prefix* instead of `yy` as the prefix for all external names produced by `yacc`. The names affected shall include the functions `yyparse()`, `yylex()`, and `yyerror()`, and the variables `yyval`, `yychar`, and `yydebug`. (In the remainder of this clause, the six symbols cited are referenced using their default names only as a notational convenience.) Local names may also be affected by the `-p` option; however, the `-p` option shall not affect `yacc`-generated `#define` symbols.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

- 968 `-t` Modify conditional compilation directives to permit compilation of
 969 debugging code in the code file. Runtime debugging statements
 970 shall be always contained in the code file, but by default condi-
 971 tional compilation directives prevent their compilation.
- 972 `-v` Write a file containing a description of the parser and a report of
 973 conflicts generated by ambiguities in the grammar.

974 **A.3.4 Operands**

975 The following operand is required:

- 976 *grammar* A pathname of a file containing instructions, hereafter called
 977 *grammar*, for which a parser is to be created. The format for the
 978 grammar is described in A.3.7.

979 **A.3.5 External Influences**

980 **A.3.5.1 Standard Input**

981 None.

982 **A.3.5.2 Input Files**

983 The file *grammar* shall be a text file formatted as specified in A.3.7.

984 **A.3.5.3 Environment Variables**

985 The following environment variables shall affect the execution of *yacc*:

- 986 **LANG** This variable shall determine the locale to use for the
 987 locale categories when both **LC_ALL** and the correspond-
 988 ing environment variable (beginning with **LC_**) do not
 989 specify a locale. See 2.6.
- 990 **LC_ALL** This variable shall determine the locale to be used to over-
 991 ride any values for locale categories specified by the set-
 992 tings of **LANG** or any environment variables beginning
 993 with **LC_**.
- 994 **LC_CTYPE** This variable shall determine the locale for the interpreta-
 995 tion of sequences of bytes of text data as characters (e.g.,
 996 single- versus multibyte characters in arguments and
 997 input files).
- 998 **LC_MESSAGES** This variable shall determine the language in which mes-
 999 sages should be written.

1000 The **LANG** and **LC_*** variables shall affect the execution of the *yacc* utility as
 1001 stated. The *main()* function defined in A.3.7.6 shall call

Copyright © 1991 IEEE. All rights reserved.
 This is an unapproved IEEE Standards Draft, subject to change.

1002 setlocale(LC_ALL, "")

1003 and thus, the program generated by `yacc` shall also be affected by the the con-
1004 tents of these variables at runtime.

1005 **A.3.5.4 Asynchronous Events**

1006 Default.

1007 **A.3.6 External Effects**

1008 **A.3.6.1 Standard Output**

1009 None.

1010 **A.3.6.2 Standard Error**

1011 If shift/reduce or reduce/reduce conflicts are detected in *grammar*, `yacc` writes a
1012 report of those conflicts to the standard error in an unspecified format.

1013 Standard error is also used for diagnostic messages.

1014 **A.3.6.3 Output Files**

1015 The code file, the header file, and the description file shall be text files. All are
1016 described in the following subclauses.

1017 **A.3.6.3.1 Code file**

1018 This file shall contain the C source code for the `yyparse()` routine. It shall contain
1019 code for the various semantic actions with macro substitution performed on them
1020 as described in A.3.7. It shall also contain a copy of the `#define` statements in 2
1021 the header file. If a `%union` declaration is used, the declaration for `YYSTYPE` 2
1022 shall be also included in this file. 2

1023 The contents of the Program Section (see A.3.7.1.4) of the input file shall then be
1024 included.

1025 **A.3.6.3.2 Header file**

1026 The header file shall contain `#define` statements that associate the token
1027 numbers with the token names. This allows source files other than the code file to
1028 access the token codes. If a `%union` declaration is used, the declaration for `YYS-`
1029 `TYPE` and an `extern YYSTYPE yylval` declaration shall be also included in this
1030 file.

1031 **A.3.6.3.3 Description file**

1032 The description file shall be a text file containing a description of the state
 1033 machine corresponding to the parser, using an unspecified format. Limits for 2
 1034 internal tables (see A.3.7.9) also shall be reported, in an implementation-defined 2
 1035 manner. 2

1036 **A.3.7 Extended Description**

1037 The `yacc` command accepts a language that is used to define a grammar for a tar-
 1038 get language to be parsed by the tables and code generated by `yacc`. The
 1039 language accepted by `yacc` as a grammar for the target language is described
 1040 below using the `yacc` input language itself.

1041 The input *grammar* includes rules describing the input structure of the target
 1042 language, and code to be invoked when these rules are recognized to provide the
 1043 associated semantic action. The code to be executed shall appear as bodies of text
 1044 that are intended to be C language code. The C language inclusions are presumed
 1045 to form a correct function when processed by `yacc` into its output files. The code
 1046 included in this way shall be executed during the recognition of the target
 1047 language.

1048 Given a grammar, the `yacc` utility generates the files described in A.3.6.3. The 2
 1049 code file can be compiled and linked using `c89`. If the declaration and programs 2
 1050 sections of the grammar file did not include definitions of `main()`, `yylex()`, and 2
 1051 `yyerror()`, the compiled output requires linking with externally supplied version of 2
 1052 those functions. Default versions of `main()` and `yyerror()` are supplied in the 2
 1053 `yacc` library and can be linked in by using the `-l y` operand to `c89`. The `yacc` 1
 1054 library interfaces need not support interfaces with other than the default `yy` sym- 1
 1055 bol prefix. The application provides the lexical analyzer function, `yylex()`; the `lex` 1
 1056 utility (see A.2) is specifically designed to generate such a routine.

2

1057 **A.3.7.1 Input Language**

1058 Every specification file shall consist of three sections: *declarations*, *grammar*
 1059 *rules*, and *programs*, separated by double percent-signs (%%). The declarations
 1060 and programs sections can be empty. If the latter is empty, the preceding %%
 1061 mark separating it from the rules section can be omitted.

1062 The input is free form text following the structure of the grammar defined below.

1063 **A.3.7.1.1 Lexical Structure of the Grammar**

1064 The characters <blank>s, <newline>s, and <form-feed>s shall be ignored,
 1065 except that they shall not appear in names or multicharacter reserved symbols.
 1066 Comments shall be enclosed in `/* ... */`, and can appear wherever a name is
 1067 valid.

Names are of arbitrary length, made up of letters, periods (.), underscores (_), and noninitial digits. Upper- and lowercase letters are distinct. Portable applications shall not use names beginning in `yy` or `YY` since the `yacc` parser uses such names. Many of the names appear in the final output of `yacc`, and thus they should be chosen to conform with any additional rules created by the C compiler to be used. In particular they will appear in `#define` statements.

A literal shall consist of a single character enclosed in single-quotes ('). All of the escape sequences supported for character constants by the C Standard {7} (3.1.3.4) shall be supported by `yacc`.

The relationship with the lexical analyzer is discussed in detail below.

The NUL character shall not be used in grammar rules or literals.

A.3.7.1.2 Declarations Section

The declarations section is used to define the symbols used to define the target language and their relationship with each other. In particular, much of the additional information required to resolve ambiguities in the context-free grammar for the target language is provided here.

Usually `yacc` assigns the relationship between the symbolic names it generates and their underlying numeric value. The declarations section makes it possible to control the assignment of these values.

It is also possible to keep semantic information associated with the tokens currently on the parse stack in a user-defined C language union, if the members of the union are associated with the various names in the grammar. The declarations section provides for this as well.

The first group of declarators below all take a list of names as arguments. That list can optionally be preceded by the name of a C union member (called a *tag* below) appearing within "<" and ">". (As an exception to the typographical conventions of the rest of this standard, in this case `<tag>` does not represent a meta-variable, but the literal angle bracket characters surrounding a symbol.) The use of *tag* specifies that the tokens named on this line are to be of the same C type as the union member referenced by *tag*. This is discussed in more detail below.

For lists used to define tokens, the first appearance of a given token can be followed by a positive integer (as a string of decimal digits). If this is done, the underlying value assigned to it for lexical purposes shall be taken to be that number.

```
%token [<tag>] name [number] [name [number]]...
```

Declares *name*(s) to be a token. If *tag* is present, the C type for all tokens on this line shall be declared to be the type referenced by *tag*. If a positive integer, *number*, follows a *name*, that value shall be assigned to the token.

```
%left [<tag>] name [number] [name [number]]...
```

```
%right [<tag>] name [number] [name [number]]...
```

Declares *name* to be a token, and assigns precedence to it. One or

more lines, each beginning with one of these symbols can appear in this section. All tokens on the same line have the same precedence level and associativity; the lines are in order of increasing precedence or binding strength. `%left` denotes that the operators on that line are left associative, and `%right` similarly denotes right associative operators. If *tag* is present, it shall declare a C type for *name(s)* as described for `%token`.

`%nonassoc` [*<tag>*] *name* [*number*] [*name* [*number*]]...

Declares *name* to be a token, and indicates that this cannot be used associatively. If the parser encounters associative use of this token it shall report an error. If *tag* is present, it shall declare a C type for *name(s)* as described for `%token`.

`%type` *<tag> name...*

Declares that union member *name(s)* are nonterminals, and thus it is required to have a *tag* field at its beginning. Because it deals with nonterminals only, assigning a token number or using a literal is also prohibited. If this construct is present, `yacc` shall perform type checking; if this construct is not present, the parse stack shall hold only the `int` type.

Every name used in *grammar* undefined by a `%token`, `%left`, `%right`, or `%nonassoc` declaration is assumed to represent a nonterminal symbol. The `yacc` utility shall report an error for any nonterminal symbol that does not appear on the left side of at least one grammar rule.

Once the type, precedence, or token number of a name is specified, it shall not be changed. If the first declaration of a token does not assign a token number, `yacc` shall assign a token number. Once this assignment is made, the token number shall not be changed by explicit assignment.

The following declarators do not follow the previous pattern.

`%start` *name*

Declares the nonterminal *name* to be the *start symbol*, which represents the largest, most general structure described by the grammar rules. By default, it is the left-hand side of the first grammar rule; this default can be overridden with this declaration.

`%union` { *body of union (in C)* }

Declares the `yacc` value stack to be a union of the various types of values desired. By default, the values returned by actions (see below) and the lexical analyzer shall be integers. The `yacc` utility keeps track of types, and shall insert corresponding union member names in order to perform strict type checking of the resulting parser.

Alternatively, given that at least one *<tag>* construct is used, the union can be declared in a header file (which shall be included in the declarations section by using an `#include` construct within

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1154 %{ and %}), and a `typedef` used to define the symbol `YYSTYPE` to
 1155 represent this union. The effect of `%union` is to provide the
 1156 declaration of `YYSTYPE` directly from the input.

1157 %{ ... %} C language declarations and definitions can appear in the
 1158 declarations section, enclosed by these marks. These statements
 1159 shall be copied into the code file, and have global scope within it
 1160 so that they can be used in the rules and program sections.

1161 The declarations section shall be terminated by the token `%%`.

1162 **A.3.7.1.3 Grammar Rules**

1163 The rules section defines the context-free grammar to be accepted by the function
 1164 `yacc` generates, and associates with those rules C language actions and addi-
 1165 tional precedence information. The grammar is described below, and a formal
 1166 definition follows.

1167 The rules section is comprised of one or more grammar rules. A grammar rule
 1168 has the form:

1169 A : BODY ;

1170 The symbol `A` represents a nonterminal name, and `BODY` represents a sequence of
 1171 zero or more *names*, *literals*, and *semantic actions* that can then be followed by
 1172 optional *precedence rules*. Only the names and literals participate in the forma-
 1173 tion of the grammar; the semantic actions and precedence rules are used in other
 1174 ways. The colon and the semicolon are `yacc` punctuation. If there are several
 1175 successive grammar rules with the same left-hand side, the vertical bar `|` can be
 1176 used to avoid rewriting the left-hand side; in this case the semicolon appears only
 1177 after the last rule. The `BODY` part can be empty (or empty of names and literals)
 1178 to indicate that the nonterminal symbol matches the empty string.

1179 The `yacc` utility assigns a unique number to each rule. Rules using the vertical
 1180 bar notation are distinct rules. The number assigned to the rule appears in the
 1181 description file.

1182 The elements comprising a `BODY` are:

1183 *name*

1184 *literal* These form the rules of the grammar: *name* is either a *token* or a
 1185 *nonterminal*; *literal* stands for itself (less the lexically required
 1186 quotation marks).

1187 *semantic action*

1188 With each grammar rule, the user can associate actions to be per-
 1189 formed each time the rule is recognized in the input process.
 1190 [Note that the word “action” can also refer to the actions of the
 1191 parser (shift, reduce, etc.).]

1192 These actions can return values and can obtain the values
 1193 returned by previous actions. These values shall be kept in
 1194 objects of type `YYSTYPE` (see `%union`). The result value of the
 1195 action shall be kept on the parse stack with the left-hand side of

the rule, to be accessed by other reductions as part of their right-hand side. By using the `<tag>` information provided in the declarations section, the code generated by `yacc` can be strictly type checked and contain arbitrary information. In addition, the lexical analyzer can provide the same kinds of values for tokens, if desired.

An action is an arbitrary C statement, and as such can do input or output, call subprograms, and alter external variables. An action is one or more C statements enclosed in curly braces `{` and `}`.

Certain pseudo-variables can be used in the action. These are macros for access to data structures known internally to `yacc`.

`$$` The value of the action can be set by assigning it to `$$`. If type checking is enabled and the type of the value to be assigned cannot be determined, a diagnostic message may be generated.

`$number`

This refers to the value returned by the component specified by the token *number* in the right side of a rule, reading from left to right; *number* can be zero or negative. If it is, it refers to the data associated with the name on the parser's stack preceding the leftmost symbol of the current rule. (That is, `$0` refers to the name immediately preceding the leftmost name in the current rule, to be found on the parser's stack, and `$-1` refers to the symbol to *its* left.) If *number* refers to an element past the current point in the rule, or beyond the bottom of the stack, the result is undefined. If type checking is enabled and the type of the value to be assigned cannot be determined, a diagnostic message may be generated.

`$<tag>number`

These correspond exactly to the corresponding symbols without the *tag* inclusion, but allow for strict type checking (and preclude unwanted type conversions). The effect is that the macro is expanded to use *tag* to select an element from the `YYSTYPE` union (using *dataname.tag*). This is particularly useful if *number* is not positive.

1
1

`$<tag>$`

This imposes on the reference the type of the union member referenced by *tag*. This construction is applicable when a reference to a left context value occurs in the grammar, and provides `yacc` with a means for selecting a type.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1240		Actions can occur in the middle of a rule as well as at the end; an	
1241		action can access values returned by actions to its left, and in	
1242		turn the value it returns can be accessed by actions to its right.	
1243		An action appearing in the middle of a rule shall be equivalent to	
1244		replacing the action with a new nonterminal symbol and adding	
1245		an empty rule with that nonterminal symbol on the left-hand	
1246		side. The semantic action associated with the new rule shall be	
1247		equivalent to the original action. The use of actions within rules	
1248		might introduce conflicts that would not otherwise exist.	
1249		By default, the value of a rule shall be the value of the first ele-	
1250		ment in it. If the first element does not have a type (particularly	
1251		in the case of a literal) and type checking is turned on by <code>%type</code>	
1252		an error message shall result.	
1253	<i>precedence</i>	The keyword <code>%prec</code> can be used to change the precedence level	1
1254		associated with a particular grammar rule. Examples of this are	1
1255		in cases where a unary and binary operator have the same sym-	1
1256		bolic representation, but need to be given different precedences,	1
1257		or where the handling of an ambiguous if-else construction is	1
1258		necessary. The reserved symbol <code>%prec</code> can appear immediately	1
1259		after the body of the grammar rule and can be followed by a token	
1260		name or a literal. It shall cause the precedence of the grammar	
1261		rule to become that of the following token name or literal. The	
1262		action for the rule as a whole can follow <code>%prec</code> .	
1263		If a program section follows, the grammar rules shall be terminated by <code>%%</code> .	1
1264	A.3.7.1.4 Programs Section		
1265		The <i>programs</i> section can include the definition of the lexical analyzer <code>yylex()</code> ,	
1266		and any other functions, for example those used in the actions specified in the	
1267		grammar rules. This is C language code, and shall be included in the code file	
1268		after the tables and code generated by <code>yacc</code> . It is unspecified whether the pro-	
1269		grams section precedes or follows the semantic actions in the output file; there-	
1270		fore, if the application contains any macro definitions and declarations intended	
1271		to apply to the code in the semantic actions, it shall place them within <code>%{ ... %}</code>	
1272		in the declarations section.	
1273	A.3.7.1.5 Input Grammar		
1274		The following input to <code>yacc</code> yields a parser for the input to <code>yacc</code> . This is to be	
1275		taken as the formal specification of the grammar of <code>yacc</code> , notwithstanding	
1276		conflicts that may appear elsewhere.	
1277		The lexical structure is defined less precisely; the previous section on A.3.7.1.1	
1278		defines most terms. The correspondence between the previous terms and the	
1279		tokens below is as follows.	
1280	IDENTIFIER	This corresponds to the concept of <i>name</i> , given previously.	
1281		It also includes literals as defined previously.	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

```

1282     C_IDENTIFIER    This is a name, and additionally it is known to be followed
1283                      by a colon. A literal cannot yield this token.

1284     NUMBER           A string of digits (a nonnegative decimal integer).

1285     TYPE
1286     LEFT
1287     MARK
1288     etc.              These correspond directly to %type, %left, %, etc.

1289     { ... }           This indicates C language source code, with the possible
1290                      inclusion of $ macros as discussed previously.

1291  /*      Grammar for the input to yacc */
1292  /*      Basic entries */
1293  /*      The following are recognized by the lexical analyzer */
1294  %token  IDENTIFIER    /* includes identifiers and literals */
1295  %token  C_IDENTIFIER  /* identifier (but not literal)
1296                      followed by a : */
1297  %token  NUMBER        /* [0-9][0-9]* */
1298  /*      Reserved words : %type=>TYPE %left=>LEFT, etc. */
1299  %token  LEFT RIGHT NONASSOC TOKEN PREC TYPE START UNION

1300  %token  MARK          /* the %% mark */
1301  %token  LCURL          /* the %{ mark */
1302  %token  RCURL          /* the }% mark */

1303  /*      8-bit character literals stand for themselves; */
1304  /*      tokens have to be defined for multibyte characters */
1305  %start  spec

1306  %%

1307  spec  : defs MARK rules tail
1308        ;

1309  tail  : MARK
1310        {
1311          /* In this action, set up the rest of the file */
1312        }
1313        | /* empty; the second MARK is optional */
1314        ;

1315  defs  : /* empty */
1316        | defs def
1317        ;

1318  def   : START IDENTIFIER
1319        | UNION
1320        {
1321          /* Copy union definition to output */

```

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.


```

1322     }
1323     | LCURL
1324     {
1325         /* Copy C code to output file */
1326     }
1327     RCURL
1328     | rword tag nlist
1329     ;

1330 rword : TOKEN
1331     | LEFT
1332     | RIGHT
1333     | NONASSOC
1334     | TYPE
1335     ;

1336 tag   : /* empty: union tag id optional */
1337     | '<' IDENTIFIER '>'
1338     ;

1339 nlist : nmno
1340     | nlist nmno
1341     ;

1342 nmno  : IDENTIFIER          /* Note: literal invalid with % type */
1343     | IDENTIFIER NUMBER    /* Note: invalid with % type */
1344     ;

1345 /*      rule section */

1346 rules : C_IDENTIFIER rbody prec
1347     | rules rule
1348     ;

1349 rule  : C_IDENTIFIER rbody prec
1350     | '|' rbody prec
1351     ;

1352 rbody : /* empty */
1353     | rbody IDENTIFIER
1354     | rbody act
1355     ;

1356 act   : '{'
1357     | {
1358         /* Copy action, translate $$, etc. */
1359     }
1360     | '}'
1361     ;

1362 prec  : /* empty */
1363     | PREC IDENTIFIER
1364     | PREC IDENTIFIER act
1365     | prec ';'
1366     ;

```

A.3.7.2 Conflicts

The parser produced for an input grammar may contain states in which conflicts occur. The conflicts occur because the grammar is not LALR(1). An ambiguous grammar always contains at least one LALR(1) conflict. The `yacc` utility shall resolve all conflicts, using either default rules or user-specified precedence rules.

Conflicts are either “shift/reduce conflicts” or “reduce/reduce conflicts.” A shift/reduce conflict is where, for a given state and lookahead symbol, both a shift action and a reduce action are possible. A reduce/reduce conflict is where, for a given state and lookahead symbol, reductions by two different rules are possible.

The rules below describe how to specify what actions to take when a conflict occurs. Not all shift/reduce conflicts can be successfully resolved this way because the conflict may be due to something other than ambiguity, so incautious use of these facilities can cause the language accepted by the parser to be much different than was intended. The description file shall contain sufficient information to understand the cause of the conflict. Where ambiguity is the reason either the default or explicit rules should be adequate to produce a working parser.

The declared precedences and associativities (see A.3.7.1.2) are used to resolve parsing conflicts as follows:

- (1) A precedence and associativity is associated with each grammar rule; it is the precedence and associativity of the last token or literal in the body of the rule. If the `%prec` keyword is used, it overrides this default. Some grammar rules might not have both precedence and associativity.
- (2) If there is a shift/reduce conflict, and both the grammar rule and the input symbol have precedence and associativity associated with them, then the conflict is resolved in favor of the action (shift or reduce) associated with the higher precedence. If the precedences are the same, then the associativity is used; left associative implies reduce, right associative implies shift, and nonassociative implies an error in the string being parsed.
- (3) When there is a shift/reduce conflict that cannot be resolved by rule (2), the shift is done. Conflicts resolved this way are counted in the diagnostic output described in A.3.7.3.
- (4) When there is a reduce/reduce conflict, a reduction is done by the grammar rule that occurs earlier in the input sequence. Conflicts resolved this way are counted in the diagnostic output described in A.3.7.3.

Conflicts resolved by precedence or associativity shall not be counted in the shift/reduce and reduce/reduce conflicts reported by `yacc` on either standard error or in the description file.

A.3.7.3 Error Handling

The token `error` shall be reserved for error handling. The name `error` can be used in grammar rules. It indicates places where the parser can recover from a syntax error. The default value of `error` shall be 256. Its value can be changed using a `%token` declaration. The lexical analyzer should not return the value of `error`.

The parser shall detect a syntax error when it is in a state where the action associated with the lookahead symbol is `error`. A semantic action can cause the parser to initiate error handling by executing the macro `YYERROR`. When `YYERROR` is executed, the semantic action shall pass control back to the parser. `YYERROR` cannot be used outside of semantic actions.

When the parser detects a syntax error, it normally calls `yyerror` with the character string `"syntax error"` as its argument. The call shall not be made if the parser is still recovering from a previous error when the error is detected. The parser is considered to be recovering from a previous error until the parser has shifted over at least three normal input symbols since the last error was detected or a semantic action has executed the macro `yyerrok`. The parser shall not call `yyerror` when `YYERROR` is executed.

The macro function `YYRECOVERING()` shall return 1 if a syntax error has been detected and the parser has not yet fully recovered from it. Otherwise, zero shall be returned.

When a syntax error is detected by the parser, the parser shall check if a previous syntax error has been detected. If a previous error was detected, and if no normal input symbols have been shifted since the preceding error was detected, the parser checks if the lookahead symbol is an endmarker (see A.3.7.4). If it is, the parser shall return with a nonzero value. Otherwise, the lookahead symbol shall be discarded and normal parsing shall resume.

When `YYERROR` is executed or when the parser detects a syntax error and no previous error has been detected, or at least one normal input symbol has been shifted since the previous error was detected, the parser shall pop back one state at a time until the parse stack is empty or the current state allows a shift over `error`. If the parser empties the parse stack, it shall return with a nonzero value. Otherwise, it shall shift over `error` and then resume normal parsing. If the parser reads a lookahead symbol before the error was detected, that symbol shall still be the lookahead symbol when parsing is resumed.

The macro `yyerrok` in a semantic action shall cause the parser to act as if it has fully recovered from any previous errors. The macro `yyclearin` shall cause the parser to discard the current lookahead token. If the current lookahead token has not yet been read, `yyclearin` shall have no effect.

The macro `YYACCEPT` shall cause the parser to return with the value zero. The macro `YYABORT` shall cause the parser to return with a nonzero value.

1446 **A.3.7.4 Interface to the Lexical Analyzer**

1447 The *yylex()* function is an integer-valued function that returns a *token number*
 1448 representing the kind of token read. If there is a value associated with the token
 1449 returned by *yylex()* (see the discussion of *tag* above), it shall be assigned to the
 1450 external variable *yylval*.

1451 If the parser and *yylex()* do not agree on these token numbers, reliable communi-
 1452 cation between them cannot occur. For (one character) literals, the token is sim-
 1453 ply the numeric value of the character in the current character set. The numbers
 1454 for other tokens can either be chosen by *yacc*, or chosen by the user. In either
 1455 case, the *#define* construct of C is used to allow *yylex()* to return these numbers
 1456 symbolically. The *#define* statements are put into the code file, and the header
 1457 file if that file is requested. The set of characters permitted by *yacc* in an
 1458 identifier is larger than that permitted by C. Token names found to contain such
 1459 characters shall not be included in the *#define* declarations.

1460 If the token numbers are chosen by *yacc*, the tokens other than literals shall be
 1461 assigned numbers greater than 256, although no order is implied. A token can be 1
 1462 explicitly assigned a number by following its first appearance in the declarations
 1463 section with a number. Names and literals not defined this way retain their
 1464 default definition. All assigned token numbers shall be unique and distinct from
 1465 the token numbers used for literals. If duplicate token numbers cause conflicts in
 1466 parser generation, *yacc* shall report an error; otherwise, it is unspecified whether
 1467 the token assignment is accepted or an error is reported.

1468 The end of the input is marked by a special token called the *endmarker*, which
 1469 has a token number that is zero or negative. (These values are invalid for any
 1470 other token.) All lexical analyzers shall return zero or negative as a token
 1471 number upon reaching the end of their input. If the tokens up to, but excluding,
 1472 the endmarker form a structure that matches the start symbol, the parser shall
 1473 accept the input. If the endmarker is seen in any other context, it shall be con-
 1474 sidered an error.

1475 **A.3.7.5 Completing the Program**

1476 In addition to *yyparse()* and *yylex()*, the functions *yyerror()* and *main()* are
 1477 required to make a complete program. The application can supply *main()* and
 1478 *yyerror()*, or those routines can be obtained from the *yacc* library.

1479 **A.3.7.6 yacc Library**

1480 The following functions appear only in the *yacc* library accessible through the
 1481 *-l y* operand to *c89*; they can therefore be redefined by a portable application:

```
1482     int main(void)                                     1
1483         This function shall call yyparse() and exit with an unspecified
1484         value. Other actions within this function are unspecified.
```

```
1485     int yyerror(const char *s)                         1
1486         This function shall write the NUL-terminated argument to
```

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1487 standard error, followed by a `<newline>`.

1488 The order of the `-l y` and `-l l` operands given to `c89` is significant; the applica-
 1489 tion shall either provide its own `main()` function or ensure that `-l y` precedes
 1490 `-l l`.

1491 **A.3.7.7 Debugging the Parser**

1492 The parser generated by `yacc` shall have diagnostic facilities in it that can be
 1493 optionally enabled at either compile time or at run time (if enabled at compile
 1494 time). The compilation of the runtime debugging code is under the control of
 1495 `YYDEBUG`, a preprocessor symbol. If `YYDEBUG` has a nonzero value, the debug-
 1496 ging code shall be included. If its value is zero, the code shall not be included.

1497 In parsers where the debugging code has been included, the external `int yyde-`
 1498 `bug` can be used to turn debugging on (with a nonzero value) and off (zero value)
 1499 at run time. The initial value of `yydebug` shall be zero.

1500 When `-t` is specified, the code file shall be built such that, if `YYDEBUG` is not
 1501 already defined at compilation time (using the `c89 -D YYDEBUG` option, for exam-
 1502 ple), `YYDEBUG` shall be set explicitly to 1. When `-t` is not specified, the code file
 1503 shall be built such that, if `YYDEBUG` is not already defined, it shall be set expli-
 1504 citly to zero.

1505 The format of the debugging output is unspecified but includes at least enough
 1506 information to determine the shift and reduce actions, and the input symbols. It
 1507 also provides information about error recovery.

1508 **A.3.7.8 Algorithms**

1509 The parser constructed by `yacc` implements an LALR(1) parsing algorithm as
 1510 documented in the literature. It is unspecified whether the parser is table-driven
 1511 or direct-coded.

1512 A parser generated by `yacc` shall never request an input symbol from `yylex()`
 1513 while in a state where the only actions other than the error action are reductions
 1514 by a single rule.

1515 The literature of parsing theory defines these concepts.

1516 **A.3.7.9 Limits**

1517 The `yacc` utility may have several internal tables. The minimum maximums for
 1518 these tables are shown in Table A-4. The exact meaning of these values is imple-
 1519 mentation defined. The implementation shall define the relationship between
 1520 these values and between them and any error messages that the implementation
 1521 may generate should it run out of space for any internal structure. An implemen-
 1522 tation may combine groups of these resources into a single pool as long as the
 1523 total available to the user does not fall below the sum of the sizes specified by this
 1524 subclause.

Table A-4 – yacc Internal Limits

Limit	Minimum Maximum	Description
{NTERMS}	126	Number of tokens.
{NNONTERM}	200	Number of nonterminals.
{NPROD}	300	Number of rules.
{NSTATES}	600	Number of states.
{MEMSIZE}	5200	Length of rules. The total length, in names (tokens and nonterminals), of all the rules of the grammar. The left-hand side is counted for each rule, even if it is not explicitly repeated, as specified in A.3.7.1.3.
{ACTSIZE}	4000	Number of actions. “Actions” here (and in the description file) refer to parser actions (shift, reduce, etc.) not to semantic actions defined in A.3.7.1.3.

A.3.8 Exit Status

The yacc utility shall exit with one of the following values:

- 0 Successful completion.
- >0 An error occurred.

A.3.9 Consequences of Errors

If any errors are encountered, the run is aborted and yacc exits with a nonzero status. Partial code files and header files may be produced. The summary information in the description file shall always be produced if the -v flag is present.

A.3.10 Rationale. (This subclause is not a part of P1003.2)

The references in the Bibliography may be helpful in constructing the parser generator. The Pennello-DeRemer {B26} paper (along with the works it references) describe a technique to generate parsers that conform to this standard. Work in this area continues to be done, so implementors should consult current literature before doing any new implementations. The original paper by Knuth {B27} is the theoretical basis for this kind of parser, but the tables it generates are impractically large for reasonable grammars, and should not be used. The “equivalent to” wording is intentional to assure that the best tables that are LALR(1) can be generated.

There has been confusion between the class of grammars, the algorithms needed to generate parsers, and the algorithms needed to parse the languages. They are all reasonably orthogonal. In particular, a parser generator that accepts the full range of LR(1) grammars need not generate a table any more complex than one that accepts SLR(1) (a relatively weak class of LR grammars) for a grammar that

happens to be SLR(1). Such an implementation need not recognize the case, either; table compression can yield the SLR(1) table (or one even smaller than that) without recognizing that the grammar is SLR(1). The speed of a LR(1) parser for any class is dependent more upon the table representation and compression (or the code generation if a direct parser is generated) than upon the class of grammar that the table generator handles.

The speed of the parser generator is somewhat dependent upon the class of grammar it handles. However, the original Knuth {B27} algorithms for constructing LR parsers was judged by its author to be impractically slow at that time. Although full LR is more complex than LALR(1), as computer speeds and algorithms improve, the difference (in terms of acceptable wall-clock execution time) is becoming less significant.

Potential authors are cautioned that the Penello-DeRemer paper previously cited identifies a bug (an oversimplification of the computation of LALR(1) lookahead sets) in some of the LALR(1) algorithm statements that preceded it to publication. They should take the time to seek out that paper, as well as current relevant work, particularly Aho's {B22}.

Examples, Usage

Access to the yacc library is obtained with library search operands to `c89`. To use the yacc library `main()`,

```
c89 y.tab.c -l y
```

Both the lex library and the yacc library contain `main()`. To access the yacc `main()`,

```
c89 y.tab.c lex.yy.c -l y -l l
```

This ensures that the yacc library is searched first, so that its `main()` is used.

The historical yacc libraries have contained two simple functions that are normally coded by the application programmer. These library functions are similar to the following code:

```

1593     #include <locale.h>
1594     int main(void)
1595     {
1596         extern int yyparse();
1597
1598         setlocale(LC_ALL, "");
1599
1600         /* If the following parser is one created by lex, the
1601            application must be careful to ensure that LC_CTYPE
1602            and LC_COLLATE are set to the POSIX Locale. */
1603         (void) yyparse();
1604         return (0);
1605     }
1606
1607     #include <stdio.h>
1608
1609     int yyerror(const char *msg)
1610     {
1611         (void) fprintf(stderr, "%s\n", msg);
1612         return (0);
1613     }

```

Historical implementations experience name conflicts on the names `yacc.tmp`, `yacc.acts`, `yacc.debug`, `y.tab.c`, `y.tab.h`, and `y.output` if more than one copy of `yacc` is running in a single directory at one time. The `-b` option was added to overcome this problem. The related problem of allowing multiple `yacc` parsers to be placed in the same file was addressed by adding a `-p` option to override the previously hardcoded `yy` variable prefix. (The `-p` option name was selected from a historical implementation.) Implementations will also have to be cognizant of 2.11.6.3, which requires that any temporary files used by `yacc` also be named to avoid collisions.

The description of the `-p` option specifies the minimal set of function and variable names that cause conflict when multiple parsers are linked together. `YYSTYPE` does not need to be changed. Instead, the programmer can use `-b` to give the header files for different parsers different names, and then the file with the `yylex()` for a given parser can include the header for that parser. Names such as `yyclearerr` don't need to be changed because they are used only in the actions; they do not have linkage. It is possible that an implementation will have other names, either internal ones for implementing things such as `yyclearerr`, or providing non-standard features, that it wants to change with `-p`.

The `-b` option was added to provide a portable method for permitting `yacc` to work on multiple separate parsers in the same directory. If a directory contains more than one `yacc` grammar, and both grammars are constructed at the same time (by, say, a parallel make program), conflict results. While the solution is not historical practice, it corrects a known deficiency in historical implementations. Corresponding changes were made to all sections that referenced the filenames `y.tab.c` (now “the code file”), `y.tab.h` (now “the header file”), and `y.output` (now “the description file”).

The grammar for `yacc` input is based on System V documentation. The textual description shows there that the `;` is required at the end of the rule. The grammar and the implementation do not require this. (The use of `C_IDENTIFIER` causes a reduce to occur in the right place.) 1

Also, in that implementation, the constructs such as `%token` can be terminated by a semicolon, but this is not permitted by the grammar. The keywords such as `%token` can also appear in uppercase, which is again not discussed. In most places where `%` is used, `\` can be substituted, and there are alternate spellings for some of the symbols (e.g. `%LEFT` can be `%<` or even `\<`). 1

Multibyte characters should be recognized by the lexical analyzer and returned as tokens. They should not be returned as multibyte character literals. The token error that is used for error recovery is normally assigned the value 256 in the historical implementation. Thus, the token value 256, which used in many multibyte character sets, is not available for use as the value of a user-defined token. 2

Historically, `<tag>` can contain any characters except `>`, including white space, in the implementation. However, since the `tag` must reference a Standard C union member, in practice conforming implementations need only support the set of characters for Standard C identifiers in this context.

Some historical implementations are known to accept actions that are terminated by a period. Historical implementations often allow `$` in names. A conforming implementation need support neither of these behaviors.

Unary operators that are the same token as a binary operator in general need their precedence adjusted. This is handled by the `%prec` advisory symbol associated with the particular grammar rule defining that unary operator. See A. Applications are not required to use this operator for unary operators, but the grammars that do not require it are rare.

Deciding when to use `%prec` illustrates the difficulty in specifying the behavior of `yacc`. There may be situations in which the *grammar* is not strictly speaking in error, and yet `yacc` cannot interpret it unambiguously. The resolution of ambiguities in the grammar can in many instances be resolved by providing additional information, such as using `%type` or `%union` declarations. It is often easier and it usually yields a smaller parser to take this alternative when it is appropriate.

The size and execution time of a program produced without the runtime debugging code is usually smaller and slightly faster in historical implementations.

There is a fair amount of material in this that appears tutorial in nature; some of it has been moved to the Rationale in Draft 9 to simplify the specification. It is hard to avoid because of the need to define terms at least informally. The alternative is to bring in one of the parser generator texts and use its terminology directly, but since there is some variation in that terminology, it was felt that informal definitions of the terms so that someone who understood the concepts would be sure to understand the terms would make the standard stand alone from any specific text.

Statistics messages from several historical implementations include the following types of information:

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1680 $n/512$ terminals, $n/300$ nonterminals
 1681 $n/600$ grammar rules, $n/1500$ states
 1682 n shift/reduce, n reduce/reduce conflicts reported
 1683 $n/350$ working sets used
 1684 memory: states, etc. $n/15000$, parser $n/15000$
 1685 $n/600$ distinct lookahead sets
 1686 n extra closures
 1687 n shift entries, n exceptions
 1688 n goto entries
 1689 n entries saved by goto default
 1690 Optimizer space used: input $n/15000$, output $n/15000$
 1691 n table entries, n zero
 1692 maximum spread: n , maximum offset: n

1693 The report of internal tables in the description file is left implementation defined 2
 1694 because all aspects of these limits are also implementation defined. Some imple- 2
 1695 mentations may use dynamic allocation techniques and have no specific limit 2
 1696 values to report. 2

1697 **History of Decisions Made**

1698 The format of the `y.output` file is not given because specification of the format
 1699 was not seen to enhance application portability. The listing is primarily intended
 1700 to help human users understand and debug the parser; use of `y.output` by a
 1701 portable application script is far-fetched. Furthermore, implementations have not
 1702 produced consistent output and no clear winner was apparent. The format
 1703 selected by the implementation should be human-readable, in addition to the
 1704 requirement that it be a text file.

1705 Standard error reports are not specifically described because they are seldom of
 1706 use to portable applications and there was no reason to restrict implementations.

1707 Some implementations recognize `= {` as equivalent to `{`, because it appears in his-
 1708 torical documentation. This construction was recognized and documented as
 1709 obsolete as long ago as 1978, in the original paper *Yacc: Yet Another Compiler-
 1710 Compiler* by Stephen C. Johnson. POSIX.2 chose to leave it as obsolete and omit
 1711 it.

Annex B (normative)

C Language Bindings Option

1 This annex describes the C language bindings to the language-independent ser-
2 vices described in Section 7.

3 The interfaces described in this annex may be provided by the conforming system;
4 however, any system claiming conformance to the Language-Independent System
5 Services C Language Bindings Option shall provide all of the interfaces described
6 here.

7 **B.0.1 C Language Bindings Option Rationale.** *(This subclause is not a part of* 8 *P1003.2)*

9 In this version of POSIX.2, the language-independent descriptions in Section 7
10 have not been developed. The language-independent syntax is being created in
11 parallel by the POSIX.1 working group. Therefore, the C language bindings
12 described in this annex are actually the full functional specifications. It is the
13 intention of the POSIX.2 working group to rectify this situation in a revision to
14 this standard, by moving the majority of the functional specifications back into
15 Section 7, leaving Annex B with only brief descriptions of the C bindings to those
16 services.

B.1 C Language Definitions

B.1.1 POSIX Symbols

Certain symbols in this annex are defined in headers. Some of those headers could also define symbols other than those defined by this standard, potentially conflicting with symbols used by the application. Also, this standard defines symbols that other standards do not permit to appear in those headers without some control on the visibility of those symbols.

Symbols called *feature test macros* are used to control the visibility of symbols that might be included in a header. Implementations, future versions of this standard, and other standards may define additional feature test macros. The `#defines` for feature test macros shall appear in the application source code before any `#include` of a header where a symbol should be visible to some, but not all, applications. If the definition of the macro does not precede the `#include`, the result is undefined.

Feature test macros shall begin with the underscore character (`_`) and an upper-case letter, or with two underscore characters.

Implementations may add symbols to the headers shown in Table B-1, provided the identifiers for those symbols begin with the corresponding reserved prefixes in Table B-1. Similarly, implementations may add symbols to the headers in Table B-1 that end in the string indicated as a reserved suffix as long as the reserved suffix is in that part of the name considered significant by the implementation. This shall be in addition to any reservations made in the C Standard [7].

After the last inclusion of a given header, an application may use any of the symbol classes reserved in Table B-1 for its own purposes, as long as the requirements in the note to Table B-1 are satisfied, noting that the symbol declared in the header may become inaccessible.

Future revisions of this standard, and other POSIX standards, are likely to use symbols in these same reserved spaces.

In addition, implementations may add members to a structure or union without controlling the visibility of those members with a feature test macro, as long as a user-defined macro with the same name cannot interfere with the correct interpretation of the program.

A conforming POSIX.2 application shall define the feature test macro in Table B-2. When an application includes a header and the `_POSIX_C_SOURCE` feature test macro is defined to be the value 1 or 2, the effect shall be the same as if `_POSIX_SOURCE` was defined as described in POSIX.1 [8].

In addition, when the application includes any of the headers defined in this standard, and `_POSIX_C_SOURCE` is defined to be the value 2:

- (1) All symbols defined in POSIX.2 to appear when the header is included shall be made visible.

Table B-1 – POSIX.2 Reserved Header Symbols

Header	Key	Reserved Prefix	Reserved Suffix
<fnmatch.h>	2	FNM_	
<glob.h>	1	gl_	
	2	GLOB_	
<limits.h>	1		_MAX
<regex.h>	1	re_	
	1	rm_	
	2	REG_	
<wordexp.h>	1	we_	
	2	WRDE_	

NOTE: The Key values are:

- (1) Prefixes and suffixes of symbols that shall not be declared or #defined by the application.
- (2) Prefixes and suffixes of symbols that shall be preceded in the application with a #undef of that symbol before any other use.

Table B-2 – _POSIX_C_SOURCE

Name	Description
_POSIX_C_SOURCE	Enable POSIX.1 {8} and POSIX.2 symbols; see text.

- (2) Symbols that are explicitly permitted, but not required, by POSIX.2 to appear in the header (including those in reserved name spaces) may be made visible.

- (3) Additional symbols shall not be made visible, unless controlled by another feature test macro.

The effect of defining the _POSIX_C_SOURCE macro to any other value is unspecified.

If there are no feature test macros present in a program, only the set of symbols defined by the C Standard {7} shall be present. For each feature test macro present, only the symbols specified by that feature test macro plus those of the C Standard {7} shall be defined when the header is included.

91 **B.1.1.1 POSIX Symbols Rationale.** *(This subclause is not a part of P1003.2)*

92 When the application defines the `_POSIX_C_SOURCE` feature test macro with 1
 93 value 2, it must be aware that all of the name space from POSIX.1 {8} and POSIX.2 1
 94 has been reserved. This does not imply that a POSIX.2 implementation must sup-
 95 port POSIX.1 {8}, just that the application must not conflict with an implementa-
 96 tion that does. The application can check `_POSIX_VERSION` and 1
 97 `_POSIX2_C_VERSION` at compile time to see which standards are supported, if 1
 98 that is necessary. This is primarily an issue for the headers `<stdio.h>`,
 99 `<limits.h>`, `<locale.h>`, and `<unistd.h>`, since other POSIX.1 {8} names
 100 appear in other headers not mentioned in POSIX.2.

101 It is expected that C bindings to future POSIX standards and revisions will define
 102 new values for `_POSIX_C_SOURCE`, with each new value reserving the name 1
 103 space for that new standard or revision, plus all earlier POSIX standards. Using a
 104 single feature test macro for all standards rather than a separate macro for each
 105 standard furthers the goal of eventually combining all of the C bindings into one
 106 standard, which will be included in an international standard that refers to a
 107 language-independent ISO/IEC 9945-1 {8}.

108 **B.1.2 Headers and Function Prototypes**

109 Implementations shall declare function prototypes for all functions. Each func-
 110 tion prototype shall appear in the header included in the synopsis of the function.

111 **B.1.3 Error Numbers**

112 Some of the functions in this annex use the variable `errno` to report errors. Such
 113 usage is documented in Errors in each specification. The usage of `errno` and the
 114 meanings of the symbolic names shall be as defined in POSIX.1 {8} B.1.3.

115 **B.1.4 C Language Definitions Rationale.** *(This subclause is not a part of P1003.2)*

116 This clause clarifies the interface to the C Standard {7}. The description was
 117 taken from POSIX.1, with one important modification. Since POSIX.1 {8} and the 1
 118 C Standard {7} were being developed and approved at about the same time, 1
 119 POSIX.1 {8} allowed “Common Usage C” implementations to give system vendors 1
 120 time to develop Standard C interfaces. Since Standard C compilers are now com- 1
 121 monly available, POSIX.2 does not explicitly describe the binding to Common 1
 122 Usage C. However, such a binding would be straightforward, as long as the rules
 123 for Common Usage C in POSIX.1 are followed.

B.2 C Numerical Limits

The following subclauses list the names of macros that C language applications can use to obtain minimum and current values for limits defined in 2.13.1.

B.2.0.1 C Numerical Limits Rationale. *(This subclause is not a part of P1003.2)*

This subclause was added in Draft 9 to give C applications access to limits at compile time. Applications can use the values from the macros without resorting to `sysconf()`. The descriptions very closely follow the descriptions of macros and limits in POSIX.1 {8}.

This definition of the limits is specific to the C language. Other language bindings might use different interfaces or names to provide equivalent information to the application.

Note that there are no C bindings or interfaces that change based on the macros in Table B-5. These macro only advertise the availability of the associated utilities.

B.2.1 C Macros for Symbolic Limits

The macros in Table B-3 shall be defined in the header `<limits.h>`. They specify values for the symbolic limits defined in 2.13.1.

Table B-3 – C Macros for Symbolic Limits

Symbolic Limit	Minimum Allowed by POSIX.2	Minimum for this Implementation
{BC_BASE_MAX}	_POSIX2_BC_BASE_MAX	BC_BASE_MAX
{BC_DIM_MAX}	_POSIX2_BC_DIM_MAX	BC_DIM_MAX
{BC_SCALE_MAX}	_POSIX2_BC_SCALE_MAX	BC_SCALE_MAX
{BC_STRING_MAX}	_POSIX2_BC_STRING_MAX	BC_STRING_MAX
{COLL_WEIGHTS_MAX}	_POSIX2_COLL_WEIGHTS_MAX	COLL_WEIGHTS_MAX
{EXPR_NEST_MAX}	_POSIX2_EXPR_NEST_MAX	EXPR_NEST_MAX
{LINE_MAX}	_POSIX2_LINE_MAX	LINE_MAX
{RE_DUP_MAX}	_POSIX2_RE_DUP_MAX	RE_DUP_MAX

The names in the first column of Table B-3 are symbolic limits as defined in 2.13.1. The names in the second column are C macros that define the smallest values permitted for the symbolic limits on any POSIX.2 implementation; they shall be defined as constant expressions with the most restrictive values specified in 2.13.1. The names in the third column are C macros that define less restrictive values provided by the implementation; each shall be defined as a constant that

- is not smaller than the associated macro in column 2, and
- is not larger than the smallest value that will be returned by `sysconf()` when the application is executed.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

163 **B.2.1.1 C Macros for Symbolic Limits Rationale.** *(This subclause is not a part of* 164 *P1003.2)*

165 The macros in column 3 of Table B-3 are required to be constant expressions.

166 If the C binding is to be used with POSIX.2 implementations over which the imple-
 167 mentor of the binding has no control, the column-3 values must be the same as
 168 column-2. If the implementation of the C binding is intended to be used with a
 169 POSIX.2 implementation that always supports a larger value than one in column
 170 2, that implementation of the binding may use the larger value for the column-3
 171 macro. If an application compiled with that binding is then used with a different
 172 POSIX.2 implementation, it is the user's fault that the application is being run in
 173 an environment in which it was not intended.

174 The application can assume, for example, that the stream created by
 175 `popen("mailx user", "w")` will accept lines of length `{LINE_MAX}`, even if this
 176 is larger than `{_POSIX2_LINE_MAX}`. However, if the application is creating a
 177 data file that might be processed on another implementation, it should use the
 178 values in column 2.

179 **B.2.2 Compile-Time Symbolic Constants for Portability Specifications**

180 The macros in Table B-4 shall be defined in the header `<unistd.h>`. These mac-
 181 ros can be used by the application, at compile time, to determine which optional
 182 facilities are present and what actions shall be taken by the implementation.

183 **Table B-4 – C Compile-Time Symbolic Constants**

184	Macro Name	Description	
185			
186	<code>_POSIX2_C_VERSION</code>	The integer value 199???.L. This value indicates the version of the inter- 187 faces in this annex that are provided by the implementation. It will 188 change with each published version of this standard to indicate the 4-digit 189 year and 2-digit month that the standard was approved by the IEEE Stan- 190 dards Board.	1 1 1 1 1
191			

192 **B.2.2.1 Compile-Time Symbolic Constants for Portability Specifications** 193 **Rationale.** *(This subclause is not a part of P1003.2)*

194 The symbolic constant `_POSIX2_C_VERSION` is analogous to `_POSIX_VERSION`,
 195 defined in POSIX.1 {8}. It indicates the version of the C interfaces that are sup-
 196 plied by the compiler and runtime library. 1

197 The version of the utilities is given by the `{POSIX2_VERSION}` limit (see 2.13.1),
 198 whose value can be obtained at runtime using `sysconf()` (see B.10.2). 1

B.2.3 Execution-Time Symbolic Constants for Portability Specifications

The macros in Table B-5 can be used by the application at execution time to determine which optional facilities are present. If a macro is defined to have the value `-1` in the header `<unistd.h>`, the implementation shall never provide that feature when the application runs under that implementation. If a macro is defined to have a value other than `-1`, the implementation shall always provide that feature. If the macro is undefined, then the `sysconf()` function (see B.10.2) can be used to determine if the feature is provided for a particular invocation of the application.

Table B-5 – C Execution-Time Symbolic Constants

Macro Name	Description
<code>_POSIX2_C_DEV</code>	The system supports the C Language Development Utilities Option (see Annex A)
<code>_POSIX2_FORT_DEV</code>	The system supports the FORTRAN Development Utilities Option (see Annex C)
<code>_POSIX2_FORT_RUN</code>	The system supports the FORTRAN Runtime Utilities Option (see Annex C)
<code>_POSIX2_LOCALEDEF</code>	The system supports the creation of locales as described in 4.35.
<code>_POSIX2_SW_DEV</code>	The system supports the Software Development Utilities Option (see Section 6)

B.2.4 POSIX.1 C Numerical Limits

The macros specified in POSIX.1 {8} to provide compile-time values for the configurable variables in Table 7-1 (see 7.8.2) shall also be visible in a POSIX.2 system. Other macros required by POSIX.1 {8} 2.9 (Numerical Limits) and 2.10 (Symbolic Constants) may also be visible in a POSIX.2 system.

B.2.4.1 POSIX.1 C Numerical Limits Rationale. *(This subclause is not a part of P1003.2)*

Subclause 7.8.2 requires that certain POSIX.1 {8} configurable variables be visible in POSIX.2. Subclause B.2.4 ensures that POSIX.2 C applications can obtain these variables using the same macros as POSIX.1 {8} C applications. It also allows an implementation to make all of the POSIX.1 {8} macros available even if `_POSIX_SOURCE` is not set. It also allows an implementation to make all of the POSIX.1 {8} symbols available even if it does not support all of POSIX.1 {8}.

B.3 C Binding for Shell Command Interface

B.3.0.1 C Binding for Shell Command Interface Rationale. *(This subclause is not a part of P1003.2)*

The *system()* and *popen()* functions should not be used by programs that have set user (or group) ID privileges, as defined in POSIX.1 {8}. The *fork()* and *exec* family of functions [except *execlp()* and *execvp()*], also defined in POSIX.1 {8}, should be used instead. This prevents any unforeseen manipulation of the user's environment that could cause execution of commands not anticipated by the calling program.

If the original and “*popen()*ed” processes both intend to read or write or read and write a common file, and either will be using FILE-type C functions [*fread()*, *fwrite()*, etc.], the rules in POSIX.1 {8} 8.2.3 must be observed.

B.3.1 C Binding for Execute Command

Function: *system()*

B.3.1.1 Synopsis

```
#include <stdlib.h>

int system(const char *command);
```

B.3.1.2 Description

This standard requires the *system()* function as described in the C Standard {7}.

The *system()* function shall execute the command specified by the string pointed to by *command*. The environment of the executed command shall be as if a child process were created using the POSIX.1 {8} *fork()* function, and the child process invoked the *sh* utility (see 4.56) using the POSIX.1 {8} *execl()* function as follows:

```
execl(<shell path>, "sh", "-c", command, (char *)0);
```

where *<shell path>* is an unspecified pathname for the *sh* utility.

The *system()* function shall ignore the SIGINT and SIGQUIT signals, and block the SIGCHLD signal, while waiting for the command to terminate. If this might cause the application to miss a signal that would have killed it, then the application should examine the return value from *system()* and take whatever action is appropriate to the application if the command terminated due to receipt of a signal.

The *system()* function shall not affect the termination status of any child of the calling processes other than the process(es) it itself creates.

The *system()* function shall not return until the child process has terminated.

B.3.1.3 Returns

If *command* is **NULL**, the *system()* function shall return nonzero.

If *command* is not **NULL**, the *system()* function shall return the termination status of the command language interpreter in the format specified by the *waitpid()* function in POSIX.1 [8]. The termination status of the command language interpreter is as specified for the *sh* utility, except that if some error prevents the command language interpreter from executing after the child process is created, the return value from *system()* shall be as if the command language interpreter had terminated using *exit*(127) or *_exit*(127). If a child process cannot be created, or if the termination status for the command language interpreter cannot be obtained, *system()* shall return -1 and set *errno* to indicate the error.

B.3.1.4 Errors

The *system()* function may set *errno* values as described by *fork()* in POSIX.1 [8].

B.3.1.5 Rationale. *(This subclause is not a part of P1003.2)*

The C Standard [7] specifies that when *command* is **NULL**, *system()* returns nonzero if there is a command interpreter available and zero if one is not available. At first reading, it might appear that POSIX.2 conflicts with this, since it requires *system(NULL)* to always return nonzero. There is no conflict, however. A POSIX.2 implementation must always have a command interpreter available, and is nonconforming if none is present. It is therefore permissible for the *system()* function on a POSIX.2 system to implement the behavior specified by the C Standard [7] as long as it is understood that the implementation is not POSIX.2 conforming if *system(NULL)* returns zero.

Note that, while *system()* must ignore SIGINT and SIGQUIT and block SIGCHLD while waiting for the child to terminate, the handling of signals in the executed command is as specified by *fork()* and *exec*. For example, if SIGINT is being caught or is set to SIG_DFL when *system()* is called, then the child will be started with SIGINT handling set to SIG_DFL.

Ignoring SIGINT and SIGQUIT in the parent process prevents coordination problems (two processes reading from the same terminal, for example) when the executed command ignores or catches one of the signals. It is also usually the correct action when the user has given a command to the application to be executed synchronously (as in the “!” command in many interactive applications). In either case, the signal should be delivered only to the child process, not to the application itself. There is one situation where ignoring the signals might have less than the desired effect. This is when the application uses *system()* to perform some task invisible to the user. If the user typed the interrupt character (^C for example) while *system()* is being used in this way, one would expect the application to be killed, but only the executed command will be killed. Applications that use *system()* in this way should carefully check the return status from *system()* to see if the executed command was successful, and should take appropriate action when the command fails.

Blocking SIGCHLD while waiting for the child to terminate prevents the application from catching the signal and obtaining status from *system()*'s child process before *system()* can get the status itself.

Examples, Usage

The context in which the utility is ultimately executed may differ from that in which the *system()* function was called. For example, file descriptors that have the FD_CLOEXEC flag set will be closed, and the process ID and parent process ID will be different. Also, if the executed utility changes its environment variables or its current working directory, that change will not be reflected in the caller's context.

Earlier drafts of this standard required, or allowed, *system()* to return with *errno* [EINTR] if it was interrupted with a signal. This error return was removed, and a requirement that *system()* not return until the child has terminated was added. This means that if a *waitpid()* call in *system()* exits with *errno* [EINTR], *system()* must re-issue the *waitpid()*. This change was made for two reasons:

- (1) There is no way for an application to clean up if *system()* returns [EINTR], short of calling *wait()*, and that could have the undesirable effect of returning status of children other than the one started by *system()*.
- (2) While it might require a change in some historical implementations, those implementations already have to be changed because they use *wait()* instead of *waitpid()*.

Note that if the application is catching SIGCHLD signals, it will receive such a signal before a successful *system()* call returns.

History of Decisions Made

The C Standard [7] requires that a call to *system()* with a **NULL** will return a nonzero value, indicating the presence of a command language interpreter available to the system. It was explicitly decided that when *command* is **NULL**, *system()* should not be required to check to make sure that the command language interpreter actually exists with the correct mode, that there are enough processes to execute it, etc. The call *system(NULL)* could, theoretically, check for such problems as too many existing child processes, and return zero. However, it would be inappropriate to return zero due to such a (presumably) transient condition. If some condition exists that is not under the control of this application and that would cause *any system()* call to fail, that system has been rendered nonconformant.

Modified in Draft 6 to reflect the availability of the *waitpid()* function in POSIX.1 [8]. To conform to this standard, *system()* must use *waitpid()*, or some similar function, instead of *wait()*.

Figure B-1 illustrates how *system()* might be implemented on a POSIX.1 [8] implementation.

```

350
351 #include <signal.h>
352 int system(const char *cmd) 1
353 {
354     int    stat;
355     pid_t  pid;
356     struct sigaction sa, savintr, savequit;
357     sigset_t saveblock;
358
359     if (cmd == NULL)
360         return(1);
361     sa.sa_handler = SIG_IGN;
362     sigemptyset(&sa.sa_mask);
363     sa.sa_flags = 0;
364     sigemptyset(&savintr.sa_mask); 1
365     sigemptyset(&savequit.sa_mask); 1
366     sigaction(SIGINT, &sa, &savintr);
367     sigaction(SIGQUIT, &sa, &savequit);
368     sigaddset(&sa.sa_mask, SIGCHLD); 1
369     sigprocmask(SIG_BLOCK, &sa.sa_mask, &saveblock);
370     if ((pid = fork()) == 0) {
371         sigaction(SIGINT, &savintr, (struct sigaction *)0);
372         sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
373         sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
374         execl("/bin/sh", "sh", "-c", cmd, (char *)0);
375         _exit(127);
376     }
377     if (pid == -1) {
378         stat = -1; /* errno comes from fork() */
379     } else {
380         while (waitpid(pid, &stat, 0) == -1) {
381             if (errno != EINTR) {
382                 stat = -1;
383                 break;
384             }
385         }
386     }
387     sigaction(SIGINT, &savintr, (struct sigaction *)0);
388     sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
389     sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
390     return(stat);
391 }

```

Figure B-1 – Sample *system()* Implementation

Note that, while a particular implementation of *system()* (such as the one above) can assume a particular path for the shell, such a path is not necessarily valid on another system. The above example is not portable, and is not intended to be. There is no defined way for an application to find the specific path for the shell. However, *confstr()* can provide a value for **PATH** that is guaranteed to find the *sh*

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

utility.

One reviewer suggested that an implementation of *system()* might want to use an environment variable such as **SHELL** to determine which command interpreter to use. The supposed implementation would use the default command interpreter if the one specified by the environment variable was not available. This would allow a user, when using an application that prompts for command lines to be processed using *system()*, to specify a different command interpreter. Such an implementation is discouraged. If the alternate command interpreter did not follow the command line syntax specified in POSIX.2, then changing **SHELL** would render *system()* nonconformant. This would affect applications that expected the specified behavior from *system()*, and since this standard does not mention that **SHELL** affects *system()*, the application would not know that it needed to unset **SHELL**.

B.3.2 C Binding for Pipe Communications with Programs

Functions: *popen()*, *pclose()*

B.3.2.1 Synopsis

```
#include <stdio.h>

FILE *popen(const char *command, const char *mode);

int pclose(FILE *stream);
```

B.3.2.2 Description

The *popen()* function shall execute the command specified by the string *command*. It shall create a pipe between the calling program and the executed command, and return a pointer to a C Standard {7} stream that can be used to either read from or write to the pipe. The *pclose()* function shall close the stream, wait for the command to terminate, and return the termination status from the command language interpreter.

The environment of the executed command shall be as if a child process were created within the *popen()* call using the *fork()* function, and the child invoked the *sh* utility using the call:

```
execl(<shell path>, "sh", "-c", command, (char *)0);
```

where *<shell path>* is an unspecified pathname for the *sh* utility. However, *popen()* shall ensure that any streams from previous *popen()* calls that remain open in the parent process are closed in the new child process.

The *mode* argument to *popen()* is a string that specifies I/O mode:

- (1) If *mode* is "r", when the child process is started its file descriptor `STDOUT_FILENO` shall be the writable end of the pipe, and the file descriptor *fileno(stream)* in the calling process, where *stream* is the stream pointer returned by *popen()*, shall be the readable end of the pipe.

(2) If *mode* is "w", when the child process is started its file descriptor `STDIN_FILENO` shall be the readable end of the pipe, and the file descriptor *fileno(stream)* in the calling process, where *stream* is the stream pointer returned by *popen()*, shall be the writable end of the pipe.

(3) If *mode* is any other value, the result is undefined.

A stream opened by *popen()* should be closed by *pclose()*. As stated above, *pclose()* shall return the termination status from the command language interpreter. However, if the application has called any of the following:

- (1) *wait()*,
- (2) *waitpid()* with a *pid* argument less than or equal to zero or equal to the process ID of the command line interpreter, or
- (3) any other function not defined in POSIX.1 {8} or POSIX.2 that could do one of the above

and one of those calls caused the termination status to be unavailable to *pclose()*, then *pclose()* shall return `-1` with *errno* set to `[ECHILD]` to report this situation. In any case, *pclose()* shall not return before the child process created by *popen()* has terminated.

If the command language interpreter cannot be executed, the child termination status returned by *pclose()* shall be as if the command language interpreter terminated using *exit(127)* or *_exit(127)*. If it can be executed, the *exit()* value shall be as described for the *sh* utility.

The *pclose()* function shall not affect the termination status of any child of the calling process other than the one created by *popen()* for the associated stream.

If the argument *stream* to *pclose()* is not a pointer to a stream created by *popen()*, the result of *pclose()* is undefined.

After *popen()*, both the parent and the child process shall be capable of executing independently before either terminates. See 2.9.1.2.

B.3.2.3 Returns

The *popen()* function shall return a **NULL** pointer if the pipe or subprocess cannot be created. Otherwise, it shall return a stream pointer as described above.

Upon successful return, *pclose()* shall return the termination status of the command language interpreter. Otherwise, *pclose()* shall return `-1` and set *errno* to indicate the error.

B.3.2.4 Errors

If any of the following conditions are detected, the *popen()* function shall return **NULL** and set *errno* to the corresponding value:

[EINVAL] The *mode* argument is invalid.

The *popen()* function may also set *errno* values as described by the POSIX.1 {8} *fork()* or *pipe()* functions.

If any of the following conditions are detected, the *pclose()* function shall return -1 and set *errno* to the corresponding value:

- [ECHILD] The status of the child process could not be obtained, as described above.

B.3.2.5 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

Because open files are shared, a mode "r" command can be used as an input filter and a mode "w" command as an output filter.

The behavior of *popen()* is specified for *modes* of "r" and "w". Other modes such as "rb" and "wb" might be supported by specific implementations, but these would not be portable features. Note that historical implementations of *popen()* only check to see if the first character of *mode* is r. Thus, a *mode* of "robert the robot" would be treated as *mode* "r", and a *mode* of "anything else" would be treated as *mode* "w".

If the application calls *waitpid()* with a *pid* argument greater than zero, and it still has a *popen()*ed stream open, it must ensure that *pid* does not refer to the process started by *popen()*.

History of Decisions Made

There is a requirement that *pclose()* not return before the child process terminates. This is intended to disallow implementations that return [EINTR] if a signal is received while waiting. If *pclose()* returned before the child terminated, there would be no way for the application to discover which child used to be associated with the stream, and it could not do the cleanup itself.

If the stream pointed to by *stream* was not created by *popen()*, historical implementations of *pclose()* return -1 without setting *errno*. To avoid requiring *pclose()* to set *errno* in this case, this standard makes the behavior undefined. An application should not use *pclose()* to close any stream that wasn't created by *popen()*.

Wording was added in Draft 10 requiring that the parent and child processes be able to execute independently. This behavior has been the intent all along, and the specific words were taken from the current draft of the POSIX.1a revision to POSIX.1 {8}. Rationale about this wording appears in B.3.1.1 of POSIX.1a.

Some historical implementations either block or ignore the signals SIGINT, SIGQUIT, and SIGHUP while waiting for the child process to terminate. Since this behavior is not described in POSIX.2, such implementations are not conforming. Also, some historical implementations return [EINTR] if a signal is received, even though the child process has not terminated. Such implementations are also considered nonconforming.

Consider, for example, an application that uses

```
popen("command", "r")
```

to start *command*, which is part of the same application. The parent writes a prompt to its standard output (presumably the terminal) and then reads from the *popened* stream. The child reads the response from the user, does some transformation on the response (pathname expansion, perhaps) and writes the result to its standard output. The parent process reads the result from the pipe, does something with it, and prints another prompt. The cycle repeats. Assuming that both processes do appropriate buffer flushing, this would be expected to work.

Modified in Draft 6 to reflect the availability of the *waitpid()* function in POSIX.1 {8}. To conform to this standard, *pclose()* must use *waitpid()*, or some similar function, instead of *wait()*.

Figure B-2 illustrates how the *pclose()* function might be implemented on a POSIX.1 {8} system.

```
int pclose(FILE *stream)
{
    int      stat;
    pid_t    pid;

    pid = <pid for process created for stream by popen()>
    (void) fclose(stream);

    while (waitpid(pid, &stat, 0) == -1) {
        if (errno != EINTR) {
            stat = -1;
            break;
        }
    }
    return(stat);
}
```

1

Figure B-2 – Sample *pclose()* Implementation

B.4 C Binding for Access Environment Variables

Function: *getenv()*

The C language binding to the service described in 7.2 shall be the POSIX.1 {8} *getenv()* function.

B.5 C Binding for Regular Expression Matching

Functions: *regcomp()*, *regexexec()*, *regfree()*, *regerror()*

B.5.1 Synopsis

```
#include <sys/types.h>
#include <regex.h>

int regcomp(regex_t *preg, const char *pattern, int cflags);

int regexexec(const regex_t *preg, const char *string,
              size_t nmatch, regmatch_t pmatch[], int eflags);

size_t regerror(int errcode, const regex_t *preg,
               char *errbuf, size_t errbuf_size);

void regfree(regex_t *preg);
```

B.5.2 Description

These functions shall interpret basic and extended regular expressions, as described in 2.8.

The header *<regex.h>* shall define the structure types *regex_t* and *regmatch_t*. The structure type *regex_t* shall include at least the member shown in Table B-6.

The structure type *regmatch_t* shall contain at least the members shown in Table B-7. The type *regoff_t*, which shall be defined in *<regex.h>*, shall be a signed arithmetic type that can hold the largest value that can be stored in either an *off_t* or a *ssize_t*.

The *regcomp()* function shall compile the regular expression contained in the string pointed to by the *pattern* argument and place the results in the structure pointed to by *preg*. The *cflags* argument shall be the bitwise inclusive OR of zero or more of the flags shown in Table B-8, which shall be defined in the header *<regex.h>*.

The default regular expression type for *pattern* shall be a Basic Regular Expression. The application can specify Extended Regular Expressions using the *REG_EXTENDED* *cflags* flag.

If the function *regcomp()* succeeds, it shall return zero; otherwise it shall return nonzero, and the content of *preg* shall be undefined.

Table B-6 – Structure Type *regex_t*

Member Type	Member Name	Description
<i>size_t</i>	<i>re_nsub</i>	Number of parenthesized subexpressions.

Table B-7 – Structure Type *regmatch_t*

Member Type	Member Name	Description	
<i>regoff_t</i>	<i>rm_so</i>	Byte offset from start of <i>string</i> to start of substring.	1
<i>regoff_t</i>	<i>rm_eo</i>	Byte offset from start of <i>string</i> of the first character after the end of substring.	1

Table B-8 – *regcomp()* *cflags* Argument

<i>flag</i>	Description
REG_EXTENDED	Use Extended Regular Expressions.
REG_ICASE	Ignore case in match. See 2.8.2.
REG_NOSUB	Report only success/fail in <i>regexexec()</i> .
REG_NEWLINE	Change the handling of <newline>, as described in the text.

Table B-9 – *regexexec()* *eflags* Argument

<i>flag</i>	Description
REG_NOTBOL	The first character of the string pointed to by <i>string</i> is not the beginning of the line. Therefore, the circumflex character (^), when taken as a special character, shall not match the beginning of <i>string</i> .
REG_NOTEOL	The last character of the string pointed to by <i>string</i> is not the end of the line. Therefore, the dollar sign (\$), when taken as a special character, shall not match the end of <i>string</i> .

If the REG_NOSUB flag was not set in *cflags*, then *regcomp()* shall set *re_nsub* to the number of parenthesized subexpressions [delimited by \ (\) in basic regular expressions or () in extended regular expressions] found in *pattern*.

The *regexexec()* function shall compare the null-terminated string specified by *string* against the compiled regular expression *preg* initialized by a previous call to *regcomp()*. If it finds a match, *regexexec()* shall return zero; otherwise it shall return nonzero indicating either no match or an error. The *eflags* argument shall be the bitwise inclusive OR of zero or more of the flags shown in Table B-9, which shall be defined in the header <regex.h>.

If *nmatch* is zero or REG_NOSUB was set in the *cflags* argument to *regcomp()*, then *regexexec()* shall ignore the *pmatch* argument. Otherwise, the *pmatch* argument shall point to an array with at least *nmatch* elements, and *regexexec()* shall fill in the elements of that array with offsets of the substrings of *string* that correspond to the parenthesized subexpressions of *pattern*: *pmatch[i].rm_so* shall be the byte offset of the beginning and *pmatch[i].rm_eo* shall be one greater than the byte offset of the end of substring *i*. (Subexpression *i* begins at the *i*th matched open parenthesis, counting from 1.) Offsets in *pmatch[0]* shall identify the substring that corresponds to the entire regular expression. Unused elements of *pmatch* up to *pmatch[nmatch-1]* shall be filled with -1. If there are more than *nmatch* subexpressions in *pattern* (*pattern* itself counts as a subexpression), then *regexexec()* shall still do the match, but shall record only the first *nmatch* substrings.

When matching a basic or extended regular expression, any given parenthesized subexpression of *pattern* might participate in the match of several different substrings of *string*, or it might not match any substring even though the pattern as a whole did match. The following rules shall be used to determine which substrings to report in *pmatch* when matching regular expressions:

- (1) If subexpression *i* in a regular expression is not contained within another subexpression, and it participated in the match several times, then the byte offsets in *pmatch[i]* shall delimit the last such match.
- (2) If subexpression *i* is not contained within another subexpression, and it did not participate in an otherwise successful match, then the byte offsets in *pmatch[i]* shall be -1. A subexpression shall not participate in the match when:
 - (a) * or \{ \} appears immediately after the subexpression in a basic regular expression, or *, ?, or { } appears immediately after the subexpression in an extended regular expression, and the subexpression did not match (matched zero times), or
 - (b) | is used in an extended regular expression to select this subexpression or another, and the other subexpression matched.
- (3) If subexpression *i* is contained within another subexpression *j*, and *i* is not contained within any other subexpression that is contained within *j*, and a match of subexpression *j* is reported in *pmatch[j]*, then the match or nonmatch of subexpression *i* reported in *pmatch[i]* shall be as described in (1) and (2) above, but within the substring reported in *pmatch[j]* rather than the whole string.
- (4) If subexpression *i* is contained in subexpression *j*, and the byte offsets in *pmatch[j]* are -1, then the byte offsets in *pmatch[i]* also shall be -1.
- (5) If subexpression *i* matched a zero-length string, then both byte offsets in *pmatch[i]* shall be the byte offset of the character or null terminator immediately following the zero-length string.

If, when *regexexec()* is called, the locale is different than when the regular expression was compiled, the result is undefined.

If REG_NEWLINE is not set in *cflags*, then a <newline> character in *pattern* or *string* shall be treated as an ordinary character. If REG_NEWLINE is set, then <newline> shall be treated as an ordinary character except as follows:

- (1) A <newline> in *string* shall not be matched by a period outside of a bracket expression (see 2.8.3.1.3) or by any form of a nonmatching list (see 2.8.3.2).
- (2) A circumflex (^) in *pattern*, when used to specify expression anchoring (see 2.8.4.4 and 2.8.4.6), shall match the zero-length string immediately after a <newline> in *string*, regardless of the setting of REG_NOTBOL.
- (3) A dollar-sign (\$) in *pattern*, when used to specify expression anchoring, shall match the zero-length string immediately before a <newline> in *string*, regardless of the setting of REG_NOTEOL.

The *regfree()* function shall free any memory allocated by *regcomp()* associated with *preg*.

The *regerror()* function provides a mapping from error codes returned by *regcomp()* and *regexexec()* to unspecified printable strings. It shall generate a string corresponding to the value of the *errcode* argument, which shall be the last nonzero value returned by *regcomp()* or *regexexec()* with the given value of *preg*. If *errcode* is not such a value, the content of the generated string is unspecified. If *preg* is *(regexexec_t)0*, but *errcode* is a value returned by a previous call to *regexexec()* or *regcomp()*, then *regerror()* still shall generate an error string corresponding to the value of *errcode*, but it might not be as detailed under some implementations.

If the *errbuf_size* argument is not zero, *regerror()* shall place the generated string into the *errbuf_size*-byte buffer pointed to by *errbuf*. If the string (including the terminating null) cannot fit in the buffer, *regerror()* shall truncate the string and null-terminate the result.

If *errbuf_size* is zero, *regerror()* shall ignore the *errbuf* argument, but shall return the integer value described below.

If the *preg* argument to *regexexec()* or *regfree()* is not a compiled regular expression returned by *regcomp()*, the result is undefined. A *preg* shall no longer be treated as a compiled regular expression after it is given to *regfree()*.

B.5.3 Returns

On successful completion, the *regcomp()* function shall return zero. On successful completion, the *regexexec()* function shall return zero to indicate that *string* matched *pattern*, or REG_NOMATCH (which shall be defined in <regex.h>) to indicate no match.

The *regerror()* function shall return the size of the buffer needed to hold the entire generated string, including the null termination. If the return value is greater than *errbuf_size*, the string returned in the buffer pointed to by *errbuf* has been truncated.

Table B-10 – *regcomp()*, *regexexec()* Return Values

Error Code	Description
REG_NOMATCH	<i>regexexec()</i> failed to match
REG_BADPAT	Invalid regular expression
REG_ECOLLATE	Invalid collating element referenced
REG_ETYPE	Invalid character class type referenced
REG_EESCAPE	Trailing \ in pattern
REG_ESUBREG	Number in \digit invalid or in error
REG_EBRACK	[] imbalance
REG_EPAREN	\(\) or () imbalance
REG_EBRACE	\{ \} imbalance
REG_BADBR	Content of \{ \} invalid: Not a number, number too large, more than two numbers, first larger than second
REG_ERANGE	Invalid endpoint in range expression
REG_ESPACE	Out of memory
REG_BADRPT	?, *, or + not preceded by valid regular expression

B.5.4 Errors

If *regcomp()* or *regexexec()* fails, it shall return a nonzero value indicating the type of failure. Table B-10 contains the names of macros for error codes that may be returned. If a code is returned, the interpretation shall be as given in the table. The implementation shall define the macros in Table B-10 in `<regex.h>`, and may define additional macros beginning with “REG_” for other error codes.

If *regcomp()* detects an illegal regular expression, it may return REG_BADPAT, or it may return one of the error codes that more precisely describes the error.

B.5.5 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

An example of using the functions is shown in Figure B-3

The following demonstrates how the REG_NOTBOL flag could be used with *regexexec()* to find all substrings in a line that match a pattern supplied by a user. (For simplicity of the example, very little error checking is done.)

```
(void) regcomp (&re, pattern, 0);
/* this call to regexexec() finds the first match on the line */
error = regexexec (&re, &buffer[0], 1, &pm, 0);
while (error == 0) {      /* while matches found */
    <substring found between pm.rm_sp and pm.rm_ep>
    /* This call to regexexec() finds the next match */
    error = regexexec (&re, pm.rm_ep, 1, &pm, REG_NOTBOL);
}
```

```

742
743 #include <regex.h>
744
745 /*
746  * Match string against the extended regular expression in
747  * pattern, treating errors as no match.
748  * Return 1 for match, 0 for no match.
749  */
750
751 int
752 match(const char *string, const char *pattern)
753 {
754     int      status;
755     regex_t re;
756
757     if (regcomp(&re, pattern, REG_EXTENDED|REG_NOSUB) != 0) {
758         return(0);          /* report error */
759     }
760     status = regexec(&re, string, (size_t) 0, NULL, 0);
761     regfree(&re);
762     if (status != 0) {
763         return(0);          /* report error */
764     }
765     return status == 0;
766

```

Figure B-3 – Example Regular Expression Matching

An application could use `regerror(code, preg, NULL, (size_t)0)` to find out how big a buffer is needed for the generated string, `malloc()` a buffer to hold the string, and then call `regerror()` again to get the string. Alternately, it could allocate a fixed, static buffer that is big enough to hold most strings (perhaps 128 bytes), and then `malloc()` a larger buffer if it finds that this is too small.

The `regmatch()` function must fill in all *nmatch* elements of *pmatch*, where *nmatch* and *pmatch* are supplied by the application, even if some elements of *pmatch* do not correspond to subexpressions in *pattern*. The application writer should note that there is probably no reason for using a value of *nmatch* that is larger than `preg->re_nsub`.

History of Decisions Made

The `REG_ICASE` flag supports the operations taken by the `grep -i` option and the historical implementations of `ex` and `vi`. Including this flag will make it easier for application code to be written that does the same thing as these utilities.

The substrings reported in *pmatch*[] are defined using offsets from the start of the string rather than pointers. Since this is a new interface, there should be no impact on historical implementations or applications, and offsets should be just as

easy to use as pointers. The change to offsets was made to facilitate future extensions in which the string to be searched is presented to *regexec()* in blocks, allowing a string to be searched that is not all in memory at once.

A new type *regoff_t* is used for the elements of *pmatch[]* to ensure that the application can represent either the largest possible array in memory (important for a POSIX.2-conforming application) or the largest possible file (important for an application using the extension where a file is searched in chunks).

The working group has rejected, at least for now, the inclusion of a *regsub()* function that would be used to do substitutions for a matched regular expression. While such a routine would be useful to some applications, its utility would be much more limited than the matching function described here. Both regular expression parsing and substitution are possible to implement without support other than that required by the C Standard [7], but matching is much more complex than substituting. The only “difficult” part of substitution, given the information supplied by *regexec()*, is finding the next character in a string when there can be multibyte characters. That is a much wider issue, and one that needs a more general solution.

The *errno* variable has not been used for error returns to avoid cluttering up the *errno* namespace for this feature.

In Draft 9, the interface was modified so that the matched substrings *rm_sp* and *rm_ep* are in a separate *regmatch_t* structure instead of in *regex_t*. This allows a single compiled regular expression to be used simultaneously in several contexts; in *main()* and a signal handler, perhaps, or in multiple threads of lightweight processes. (The *preg* argument to *regexec()* is declared with type *const*, so the implementation is not permitted to use the structure to store intermediate results.) It also allows an application to request an arbitrary number of substrings from a regular expression. (Previous versions reported only ten substrings.) The number of subexpressions in the regular expression is reported in *re_nsub* in *preg*. With this change to *regexec()*, consideration was given to dropping the REG_NOSUB flag, since the user can now specify this with a zero *nmatch* argument to *regexec()*. However, keeping REG_NOSUB allows an implementation to use a different (perhaps more efficient) algorithm if it knows in *regcomp()* that no subexpressions need be reported. The implementation is only required to fill in *pmatch* if *nmatch* is not zero and if REG_NOSUB is not specified. Note that the *size_t* type, as defined in the C Standard [7], is unsigned, so the description of *regexec()* does not need to address negative values of *nmatch*.

The rules for reporting substrings of extended regular expressions are consistent with those used by Henry Spencer’s “almost public domain” version of *regexec()*.

The REG_NOTBOL and REG_NOTEOL flags were added to *regexec()* in Draft 9. REG_NOTBOL was added to allow an application to do repeated searches for the same pattern in a line. If the pattern contains a circumflex character that should match the beginning of a line, then the pattern should only match when matched against the beginning of the line. Without the REG_NOTBOL flag, the application could rewrite the expression for subsequent matches, but in the general case this would require parsing the expression. The need for REG_NOTEOL is not as clear;

it was added for symmetry.

The addition of the *regerror()* function addresses the historical need for portable application programs to have access to error information more than “Function failed to compile/match your regular expression for unknown reasons.”

This interface provides for two different methods of dealing with error conditions. The specific error codes (REG_EBRACE, for example), defined in `<regex.h>`, allow an application to recover from an error if it is so able. Many applications, especially those that use patterns supplied by a user, will not try to deal with specific error cases, but will just use *regerror()* to obtain a human-readable error message to present to the user.

The *regerror()* function uses a scheme similar to *confstr()* to deal with the problem of allocating memory to hold the generated string. The scheme used by *strerror()* in the C Standard [7] was considered unacceptable since it creates difficulties for multithreaded applications. (POSIX.4a, a standard for threads, started balloting in January 1991.) A different scheme used by *regerror()* in one draft of this standard was eliminated to improve internal consistency, and because the current interface produced greater consensus than the other.

The *preg* argument is provided to *regerror()* to allow an implementation to generate a more descriptive message than would be possible with *errcode* alone. An implementation might, for example, save the character offset of the offending character of the pattern in a field of *preg*, and then include that in the generated message string. The implementation may also ignore *preg*.

A REG_FILENAME flag was considered, but omitted. This flag caused *regexexec()* to match patterns as described in 3.13 instead of regular expressions. This service is now provided by the *fnmatch()* function [see B.6].

B.6 C Binding for Match Filename or Pathname

Function: *fnmatch()*

B.6.1 Synopsis

```
#include <fnmatch.h>

int fnmatch(const char *pattern, const char *string, int flags);
```

B.6.2 Description

The *fnmatch()* function shall match patterns as described in 3.13.1 and 3.13.2. It checks the string specified by the *string* argument to see if it matches the pattern specified by the *pattern* argument.

The *flags* argument modifies the interpretation of *pattern* and *string*. It is the bit-wise inclusive OR of zero or more of the flags shown in Table B-11, which are defined in the header *<fnmatch.h>*. If the FNM_PATHNAME flag is set in *flags*, then a slash character in *string* shall be explicitly matched by a slash in *pattern*; it shall not be matched by either the asterisk or question-mark special characters, nor by a bracket expression. If the FNM_PATHNAME flag is not set, the slash character shall be treated as an ordinary character.

Table B-11 – *fnmatch()* *flags* Argument

<i>flags</i>	Description	
FNM_NOESCAPE	Disable backslash escaping	1
FNM_PATHNAME	Slash in <i>string</i> only matches slash in <i>pattern</i>	
FNM_PERIOD	Leading period in <i>string</i> must be exactly matched by period in <i>pattern</i>	

If FNM_NOESCAPE is not set in *flags*, a backslash character (\) in *pattern* followed by any other character shall match that second character in *string*. In particular, '\\\' shall match a backslash in *string*. If FNM_NOESCAPE is set, a backslash character shall be treated as an ordinary character.

If FNM_PERIOD is set in *flags*, then a leading period in *string* shall match a period in *pattern* as described by rule (2) in 3.13.2, where the location of “leading” is indicated by the value of FNM_PATHNAME:

- If FNM_PATHNAME is set, a period is “leading” if it is the first character in *string* or if it immediately follows a slash.
- If FNM_PATHNAME is not set, a period is “leading” only if it is the first character of *string*.

If FNM_PERIOD is not set, then no special restrictions shall be placed on matching a period.

890 **B.6.3 Returns**

891 If *string* matches the pattern specified by *pattern*, then *fnmatch()* shall return
 892 zero. If there is no match, *fnmatch()* shall return FNM_NOMATCH, which shall be
 893 defined in the header `<fnmatch.h>`. If an error occurs, *fnmatch()* shall return
 894 another nonzero value.

895 **B.6.4 Errors**

896 This standard does not specify any error conditions that are required to be
 897 detected by the *fnmatch()* function. Some errors may be detected under
 898 unspecified conditions.

899 **B.6.5 Rationale.** (*This subclause is not a part of P1003.2*)

900 **Examples, Usage**

901 The *fnmatch()* function has two major uses. It could be used by an application or
 902 utility that needs to read a directory and apply a pattern against each entry. The
 903 *find* utility is an example of this. It can also be used by the *pax* utility to process
 904 its *pattern* operands, or by applications that need to match strings in a similar
 905 manner.

906 **History of Decisions Made**

907 This function replaces the REG_FILENAME flag of *regcomp()* in early drafts. It
 908 provides virtually the same functionality as the *regcomp()* and *regexexec()* functions
 909 using the REG_FILENAME and REG_FSLASH flags [the REG_FSLASH flag was pro-
 910 posed for *regcomp()*, and would have had the opposite effect from
 911 FMN_PATHNAME], but with a simpler interface and less overhead.

912 The name *fnmatch()* is intended to imply *filename* match, rather than *pathname*
 913 match. The default action of this function is to match filenames, rather than
 914 pathnames, since it gives no special significance to the slash character. With the
 915 FNM_PATHNAME flag, *fnmatch()* does match pathnames, but without tilde expan-
 916 sion, parameter expansion, or special treatment for period at the beginning of a
 917 filename.

B.7 C Binding for Command Option Parsing

Function: *getopt()*

B.7.1 Synopsis

```
#include <unistd.h>
int getopt(int argc, char * const argv[], const char *optstring);      1
extern char *optarg;
extern int optind, opterr, optopt;
```

B.7.2 Description

The *getopt()* function is a command-line parser that can be used by applications that follow Utility Syntax Guidelines 3, 4, 5, 6, 7, 9, and 10 in 2.10.2. The remaining guidelines are not addressed by *getopt()* and are the responsibility of the application.

The parameters *argc* and *argv* are the argument count and argument array as passed to *main()*. The argument *optstring* is a string of recognized option characters; if a character is followed by a colon, the option takes an argument. All option characters allowed by Utility Syntax Guideline 3 are allowed in *optstring*. The implementation may accept other characters as an extension.

The variable *optind* is the index of the next element of the *argv*[] vector to be processed. It is initialized to 1 by the system, and *getopt()* updates it when it finishes with each element of *argv* []. When an element of *argv* [] contains multiple option characters, it is unspecified how *getopt()* determines which options have already been processed.

The *getopt()* function shall return the next option character from *argv* that matches a character in *optstring*, if there is one that matches. If the option takes an argument, *getopt()* shall set the variable *optarg* to point to the option-argument as follows: 1

- (1) If the option was the last character in the string pointed to by an element of *argv*, then *optarg* contains the next element of *argv*, and *optind* shall be incremented by 2. If the resulting value of *optind* is not less than *argc*, this indicates a missing option argument, and *getopt()* shall return an error indication.
- (2) Otherwise, *optarg* points to the string following the option character in that element of *argv*, and *optind* shall be incremented by 1.

If, when *getopt()* is called, *argv[optind]* is **NULL**, **argv[optind]* is not the character `-`, or *argv[optind]* points to the string `"-"`, *getopt()* shall return `-1` without changing *optind*. If *argv[optind]* points to the string `"--"`, *getopt()* shall return `-1` after incrementing *optind*.

If *getopt()* encounters an option character that is not contained in *optstring*, it shall return the question-mark (?) character. If it detects a missing option argument, it shall return the colon character (:) if the first character of *optstring* was a colon, or a question-mark character otherwise. In either case, *getopt()* shall set the variable *optopt* to the option character that caused the error. If the application has not set the variable *opterr* to zero and the first character of *optstring* is not a colon, *getopt()* shall also print a diagnostic message to standard error using the formatting rules specified for the *getopts* utility (see 4.27.6.2).

B.7.3 Returns

The *getopt()* function shall return the next option character specified on the command line. The value *-1* shall be returned when all command line options have been parsed.

B.7.4 Errors

If an invalid option is encountered, *getopt()* shall return a question-mark character. If an option with a missing option argument is encountered, *getopt()* shall return either a question-mark or a colon, as described previously.

B.7.5 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The *getopt()* function is only required to support option characters included in Guideline 3. Many historical implementations of *getopt()* support other characters as options. This is an allowed extension, but applications that use extensions are not maximally portable. Note that support for multibyte option characters is only possible when such characters can be represented as type *int*.

The code fragment in Figure B-4 shows how one might process the arguments for a utility that can take the mutually exclusive options *a* and *b* and the options *f* and *o*, both of which require arguments.

The code in Figure B-4 accepts any of the following as equivalent:

```
cmd -ao arg path path
cmd -a -o arg path path
cmd -o arg -a path path
cmd -a -o arg -- path path
cmd -a -oarg path path
cmd -aoarg path path
```

History of Decisions Made

Support for the *optopt* variable was added in Draft 9. This documents historical practice, and allows the application to obtain the identity of the invalid option.

```

991
992 #include <unistd.h>
993 int main (int argc, char *argv[ ])
994 {
995     int c, bflg, aflg, errflg = 0;
996     char *ifile, *ofile;
997     extern char *optarg;
998     extern int optind, optopt;
999     . . .
1000     while ((c = getopt(argc, argv, ":abf:o:")) != -1) {
1001         switch (c) {
1002             case 'a':
1003                 if (bflg)
1004                     errflg = 1;
1005                 else
1006                     aflg = 1;
1007                 break;
1008             case 'b':
1009                 if (aflg)
1010                     errflg = 1;
1011                 else
1012                     bflg = 1;
1013                 bproc( );
1014                 break;
1015             case 'f':
1016                 ifile = optarg;
1017                 break;
1018             case 'o':
1019                 ofile = optarg;
1020                 break;
1021             case ':': /* -f or -o without option-arg */
1022                 fprintf (stderr,
1023                     "Option -%c requires an option-argument\n",
1024                     optopt);
1025                 errflg = 1;
1026                 break;
1027             case '?':
1028                 fprintf (stderr,
1029                     "Unrecognized option: -%c\n", optopt);
1030                 errflg = 1;
1031                 break;
1032             }
1033         }
1034         if (errflg) {
1035             fprintf(stderr, "usage: . . . ");
1036             exit(2);
1037         }
1038         for ( ; optind < argc; optind++) {
1039             if (access(argv[optind], R_OK)) {
1040                 . . .
1041             }
1042         }

```

Figure B-4 – Argument Processing with *getopt()*

The description was extensively rewritten in Draft 9 to be more explicit about how *optarg* and *optind* are set, and to recognize that this routine deals with a vector of string pointers, not directly with a shell command line.

The description was modified in Draft 9 to make it clear that *getopt()*, like the *getopts* utility, shall deal with option-arguments whether separated from the option by <blank>s or not. Note that the requirements on *getopt()* and *getopts* are more stringent than the Utility Syntax Guidelines.

The *getopt()* function has been changed to return *-1*, rather than *EOF*, so that <stdio.h> is not required.

The special significance of a colon as the first character of *optstring* was added in Draft 11 to make *getopt()* consistent with the *getopts* utility. It allows an application to make a distinction between a missing argument and an incorrect option letter without having to examine the option letter. It is true that a missing argument can only be detected in one case, but that is a case that has to be considered.

B.8 C Binding for Generate Pathnames Matching a Pattern

Functions: *glob()*, *globfree()*

B.8.1 Synopsis

```
#include <glob.h>

int glob(const char *pattern, int flags,
         int (*errfunc)(const char *epath, int eerrno), glob_t *pglob);

void globfree(glob_t *pglob);
```

B.8.2 Description

The *glob()* function is a pathname generator that implements the rules defined in 3.13, with optional support for rule (3) in 3.13.3.

The header <glob.h> defines the structure type *glob_t*, which includes at least the members shown in Table B-12.

The argument *pattern* is a pointer to a pathname pattern to be expanded. The *glob()* function shall match all accessible pathnames against this pattern and develop a list of all pathnames that match. In order to have access to a pathname, *glob()* requires search permission on every component of a path except the last and read permission on each directory of any filename component of *pattern* that contains any of the special characters ***, *?* or *[*. The *glob()* function stores the number of matched pathnames into *pglob->gl_pathc* and a pointer to a list of pointers to pathnames into *pglob->gl_pathv*. The pathnames are in sort order as defined by 2.2.2.30. The first pointer after the last pathname shall be **NULL**. If the pattern does not match any pathnames, the returned number of matched paths is set to zero.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

Table B-12 – Structure Type *glob_t*

Member Type	Member Name	Description	
<i>size_t</i>	<i>gl_pathc</i>	Count of paths matched by <i>pattern</i> .	1
<i>char **</i>	<i>gl_pathv</i>	Pointer to a list of matched pathnames.	
<i>size_t</i>	<i>gl_offs</i>	Slots to reserve at the beginning of <i>gl_pathv</i> .	1

It is the caller's responsibility to create the structure pointed to by *pglob*. The *glob()* function shall allocate other space as needed, including the memory pointed to by *gl_pathv*. The *globfree()* function shall free any space associated with *pglob* from a previous call to *glob()*.

The argument *flags* is used to control the behavior of *glob()*. The value of *flags* is the bitwise inclusive OR of any of the constants shown in Table B-13, which are defined in `<glob.h>`.

Table B-13 – *glob()* *flags* Argument

Name	Description	
GLOB_APPEND	Append pathnames generated to the ones from a previous call to <i>glob()</i> .	
GLOB_DOOFFS	Make use of <i>pglob->gl_offs</i> . If this flag is set, <i>pglob->gl_offs</i> is used to specify how many NULL pointers to add to the beginning of <i>pglob->gl_pathv</i> . In other words, <i>pglob->gl_pathv</i> shall point to <i>pglob->gl_offs</i> NULL pointers, followed by <i>pglob->gl_pathc</i> pathname pointers, followed by a NULL pointer.	
GLOB_ERR	Causes <i>glob()</i> to return when it encounters a directory that it cannot open or read. Ordinarily, <i>glob()</i> continues to find matches.	
GLOB_MARK	Each pathname that is a directory that matches <i>pattern</i> has a slash appended.	
GLOB_NOCHECK	Support rule (3) in 3.13.3. If <i>pattern</i> does not match any pathname, then <i>glob()</i> shall return a list consisting of only <i>pattern</i> , and the number of matched pathnames is 1.	
GLOB_NOESCAPE	Disable backslash escaping.	1
GLOB_NOSORT	Ordinarily, <i>glob()</i> sorts the matching pathnames according to the definition of <i>collation sequence</i> in 2.2.2.30. When this flag is used the order of pathnames returned is unspecified.	

The GLOB_APPEND flag can be used to append a new set of words to those generated by a previous call to *glob()*. The following rules apply when two or more calls to *glob()* are made with the same value of *pglob* and without intervening calls to *globfree()*:

- (1) The first such call shall not set GLOB_APPEND. All subsequent calls shall set it.

- 1123 (2) All of the calls shall set `GLOB_DOOFFS`, or all shall not set it. 1
- 1124 (3) After the second call, `pglob->gl_pathv` shall point to a list containing the
 1125 following:
- 1126 (a) Zero or more **NULLs**, as specified by `GLOB_DOOFFS` and
 1127 `pglob->gl_offs`.
- 1128 (b) Pointers to the pathnames that were in the `pglob->gl_pathv` list
 1129 before the call, in the same order as before.
- 1130 (c) Pointers to the new pathnames generated by the second call, in the
 1131 specified order.
- 1132 (4) The count returned in `pglob->gl_pathc` shall be the total number of path-
 1133 names from the two calls.

1134 The application can change any of the fields in Table B-12 after a call to `glob()`, 1
 1135 but if it does it shall reset them to the original value before a subsequent call, 1
 1136 using the same `pglob` value, to `globfree()` or `glob()` with the `GLOB_APPEND` flag. 1

1137 If, during the search, a directory is encountered that cannot be opened or read
 1138 and `errfunc` is not **NULL**, `glob()` shall call `(*errfunc)()` with two arguments:

- 1139 (1) The `epath` argument is a pointer to the path that failed.
- 1140 (2) The `eerrno` argument is the value of `errno` from the failure, as set by the
 1141 POSIX.1 {8} `opendir()`, `readdir()`, or `stat()` functions. (Other values may
 1142 be used to report other errors not explicitly documented for those func-
 1143 tions.)

1144 If `(*errfunc)()` is called and returns nonzero, or if the `GLOB_ERR` flag is set in
 1145 `flags`, `glob()` shall stop the scan and return `GLOB_ABORTED` after setting `gl_pathc`
 1146 and `gl_pathv` in `pglob` to reflect the paths already scanned. If `GLOB_ERR` is not
 1147 set and either `errfunc` is **NULL** or `(*errfunc)()` returns zero, the error shall be
 1148 ignored.

1149 B.8.3 Returns

1150 On successful completion, `glob()` shall return zero. The argument `pglob->gl_pathc`
 1151 shall return the number of matched pathnames and the argument
 1152 `pglob->gl_pathv` shall contain a pointer to a null-terminated list of matched and
 1153 sorted pathnames. However, if `pglob->gl_pathc` is zero, the content of
 1154 `pglob->gl_pathv` is undefined.

1155 B.8.4 Errors

1156 If `glob()` terminates due to an error, it shall return one of the nonzero constants
 1157 shown in Table B-14, which are defined in `<glob.h>`. The arguments
 1158 `pglob->gl_pathc` and `pglob->gl_pathv` are still set as defined above in Returns.

Table B-14 – *glob()* Error Return Values

Name	Description	
GLOB_ABORTED	The scan was stopped because GLOB_ERR was set or (<i>*errfunc</i> ()) returned nonzero.	
GLOB_NOMATCH	The <i>pattern</i> does not match any existing pathname, and GLOB_NOCHECK was not set in <i>flags</i> .	1
GLOB_NOSPACE	An attempt to allocate memory failed.	1

B.8.5 Rationale. (*This subclause is not a part of P1003.2*)**Examples, Usage**

This function is not provided for the purpose of enabling utilities to perform pathname expansion on their arguments, as this operation is performed by the shell, and utilities are explicitly not expected to redo this. Instead, it is provided for applications that need to do pathname expansion on strings obtained from other sources, such as a pattern typed by a user or read from a file.

If a utility needs to see if a pathname matches a given pattern, it can use *fnmatch()*.

Note that *gl_pathc* and *gl_pathv* have meaning even if *glob()* fails. This allows *glob()* to report partial results in the event of an error. However, if *gl_pathc* is zero, *gl_pathv* is unspecified even if *glob()* did not return an error.

The GLOB_NOCHECK option could be used when an application wants to expand a pathname if wildcards are specified, but wants to treat the pattern as just a string otherwise. The *sh* utility might use this for option-arguments, for example.

One use of the GLOB_DOOFFS flag is by applications that build an argument list for use with the POSIX.1 {8} *execv()*, *execve()*, or *execvp()* functions. Suppose, for example, that an application wants to do the equivalent of *ls -l *.c*, but for some reason *system("ls -l *.c")* is not acceptable. The application could obtain (*approximately*) the same result using the sequence:

```

globbuf.gl_offs = 2;
glob ("*.c", GLOB_DOOFFS, NULL, &globbuf);
globbuf.gl_pathv[0] = "ls";
globbuf.gl_pathv[1] = "-l";
execvp ("ls", &globbuf.gl_pathv[0]);

```

Using the same example, *ls -l *.c *.h* could be approximately simulated using GLOB_APPEND as follows:

```

globbuf.gl_offs = 2;
glob ("*.c", GLOB_DOOFFS, NULL, &globbuf);
glob ("*.h", GLOB_DOOFFS|GLOB_APPEND, NULL, &globbuf);
... etc. ...

```

The new pathnames generated by a subsequent call with GLOB_APPEND are not sorted together with the previous pathnames. This mirrors the way that the shell handles pathname expansion when multiple expansions are done on a command line.

History of Decisions Made

The interface was simplified to a useful, but less complex, subset. The *errfunc* argument was added to allow errors to be reported.

A reviewer claimed that the GLOB_DOOFFS flag is unnecessary because it could be simulated using:

```
new = (char **)malloc((n + pglob->gl_pathc + 1)
                      * sizeof (char *));
(void) memcpy (new+n, pglob->gl_pathv,
              pglob->gl_pathc * sizeof(char *));
(void) memset (new, 0, n * sizeof (char *));
free (pglob->gl_pathv);
pglob->gl_pathv = new;
```

However, this assumes that the memory pointed to by *gl_pathv* is a block that was separately created using *malloc()*. This is not necessarily the case. An application should make no assumptions about how the memory referenced by fields in *pglob* was allocated. It might have been obtained from *malloc()* in a large chunk, and then carved up within *glob()*, or it might have been created using a different memory allocator. It is not the intent of this standard to specify or imply how the memory used by *glob()* is managed.

The structure elements *gl_pathc* and *gl_pathv* were renamed from *gl_argc* and *gl_argv* in Draft 9. The old names implied an association with the parameters to *main()* that does not necessarily exist.

The GLOB_APPEND flag was added in Draft 9 at the request of a reviewer. This flag would be used when an application wants to expand several different patterns into a single list.

Tilde and parameter expansion were removed from *glob()* in Draft 9. Applications that need these expansions should use the *wordexp()* function [see B.9].

B.9 C Binding for Perform Word Expansions

Functions: *wordexp()*, *wordfree()*

B.9.1 Synopsis

```
#include <wordexp.h>

int wordexp(const char *words, wordexp_t *pwordexp, int flags);

void wordfree(wordexp_t *pwordexp);
```

B.9.2 Description

The *wordexp()* function shall perform word expansions as described in 3.6, subject to quoting as in 3.2, and place the list of expanded words into *pwordexp*. The expansions shall be the same as would be performed by the shell if *words* were the part of a command line representing the arguments to a utility. Therefore, *words* shall not contain an unquoted <newline> or any of the unquoted shell special characters |, &, ;, <, or >, except in the context of command substitution as specified in 3.6.3. It also shall not contain unquoted parentheses or braces, except in the context of command or variable substitution. If *words* contains an unquoted comment character (number sign) that is the beginning of a token, *wordexp()* may treat the comment character as a regular character, or may interpret it as a comment indicator and ignore the remainder of *words*.

The header <wordexp.h> defines the structure type *wordexp_t*, which includes at least the members shown in Table B-15.

Table B-15 – Structure Type *wordexp_t*

Member Type	Member Name	Description	
<i>size_t</i>	<i>we_wordc</i>	Count of words matched by <i>words</i> .	1
<i>char **</i>	<i>we_wordv</i>	Pointer to list of expanded words.	
<i>size_t</i>	<i>we_offs</i>	Slots to reserve at the beginning of <i>we_wordv</i> .	1

The argument *words* is a pointer to a string containing one or more words to be expanded. The *wordexp()* function shall store the number of generated words into *we_wordc* and a pointer to a list of pointers to words in *we_wordv*. Each individual field created during field splitting (see 3.6.5) or pathname expansion (see 3.6.6) is a separate word in the *we_wordv* list. The words are in order as described in 3.6. The first pointer after the last word pointer shall be **NULL**. The expansion of special parameters described in 3.5.2 is unspecified.

It is the caller's responsibility to create the structure pointed to by *pwordexp*. The *wordexp()* function allocates other space as needed, including memory pointed to

1267 by *we_wordv*. The *wordfree()* function shall free any memory associated with
 1268 *pwordexp* from a previous call to *wordexp()*.

1269 The argument *flags* is used to control the behavior of *wordexp()*. The value of
 1270 *flags* is the bitwise inclusive OR of any of the constants in Table B-16, which are
 1271 defined in `<wordexp.h>`.

1272 **Table B-16 – *wordexp()* flags Argument**

1273	Name	Description
1274		
1275	WRDE_APPEND	Append words generated to the ones from a previous call to <i>wordexp()</i> .
1276	WRDE_DOOFFS	Make use of <i>we_offs</i> . If this flag is set, <i>we_offs</i> is used to specify how many 1277 NULL pointers to add to the beginning of <i>we_wordv</i> . In other words, 1278 <i>we_wordv</i> shall point to <i>we_offs</i> NULL pointers, followed by <i>we_wordc</i> 1279 word pointers, followed by a NULL pointer.
1280	WRDE_NOCMD	Fail if command substitution, as specified in 3.6.3, is requested.
1281	WRDE_REUSE	The <i>pwordexp</i> argument was passed to a previous successful call to <i>word-</i> 1282 <i>exp()</i> , and has not been passed to <i>wordfree()</i> . The result shall be the 1283 same as if the application had called <i>wordfree()</i> and then called <i>wordexp()</i> 1284 without WRDE_REUSE.
1285	WRDE_SHOWERR	Do not redirect standard error to <code>/dev/null</code> .
1286	WRDE_UNDEF	Report error on an attempt to expand an undefined shell variable.
1287		

1288 The WRDE_APPEND flag can be used to append a new set of words to those gen-
 1289 erated by a previous call to *wordexp()*. The following rules apply when two or
 1290 more calls to *wordexp()* are made with the same value of *pwordexp* and without
 1291 intervening calls to *wordfree()*:

- 1292 (1) The first such call shall not set WRDE_APPEND. All subsequent calls
 1293 shall set it.
- 1294 (2) All of the calls shall set WRDE_DOOFFS, or all shall not set it.
- 1295 (3) After the second and each subsequent call, *we_wordv* shall point to a list
 1296 containing the following:
 - 1297 (a) Zero or more NULLs, as specified by WRDE_DOOFFS and *we_offs*.
 - 1298 (b) Pointers to the words that were in the *we_wordv* list before the call,
 1299 in the same order as before.
 - 1300 (c) Pointers to the new words generated by the latest call, in the
 1301 specified order.
- 1302 (4) The count returned in *we_wordc* shall be the total number of words from
 1303 all of the calls.

1304 The application can change any of the fields in Table B-15 after a call to *word-* 1
 1305 *exp()*, but if it does it shall reset them to the original value before a subsequent 1
 1306 call, using the same *pwordexp* value, to *wordfree()* or *wordexp()* with the 1
 1307 WRDE_APPEND or WRDE_REUSE flag. 1

1308 If *words* contains an unquoted <newline>, |, &, ;, <, >, parenthesis, or brace in
 1309 an inappropriate context, *wordexp()* shall fail, and the number of expanded words
 1310 shall be zero.

1311 Unless WRDE_SHOWERR is set in *flags*, *wordexp()* shall redirect standard error to
 1312 /dev/null for any utilities executed as a result of command substitution while
 1313 expanding *words*. If WRDE_SHOWERR is set, *wordexp()* may write messages to
 1314 standard error if syntax errors are detected while expanding *words*.

1315 If WRDE_DOOFFS is set, then *we_offs* shall have the same value for each *wor-* 1
 1316 *dexp()* call and the *wordfree()* call using a given *pglob*. 1

1317 B.9.3 Returns

1318 If no errors are encountered while expanding *words*, *wordexp()* shall return zero.
 1319 Otherwise it shall return a nonzero value.

1320 B.9.4 Errors

1321 **Table B-17 – *wordexp()* Return Values**

1322	Name	Description
1323		
1324	WRDE_BADCHAR	One of the unquoted characters , &, ;, <, >, parentheses, or braces appears
1325		in <i>words</i> in an inappropriate context.
1326	WRDE_BADVAL	Reference to undefined shell variable when WRDE_UNDEF is set in <i>flags</i> .
1327	WRDE_CMDSUB	Command substitution requested when WRDE_NOCMD was set in <i>flags</i> .
1328	WRDE_NOSPACE	Attempt to allocate memory failed
1329	WRDE_SYNTAX	Shell syntax error, such as unbalanced parentheses or unterminated
1330		string.
1331		

1332 If *wordexp()* terminates due to an error, it shall return one of the nonzero con-
 1333 stants shown in Table B-17, which shall be defined in <wordexp.h>. The imple-
 1334 mentation may define additional error returns beginning with WRDE_.

1335 If *wordexp()* returns the error value WRDE_NOSPACE, then *pwordexp->we_wordc*
 1336 and *pwordexp->we_wordv* shall be updated to reflect any words that were success-
 1337 fully expanded. In other cases, they shall not be modified.

1338 **B.9.5 Rationale.** (*This subclause is not a part of P1003.2*)

1339 **Examples, Usage**

1340 This function is intended to be used by an application that wants to do all of the
 1341 shell's expansions on a word or words obtained from a user. For example, if the
 1342 application prompts for a file name (or list of file names) and then used *wordexp()*
 1343 to process the input, the user could respond with anything that would be valid as
 1344 input to the shell.

1345 The WRDE_NOCMD flag is provided for applications that, for security or other rea-
 1346 sons, want to prevent a user from executing shell commands. Disallowing
 1347 unquoted shell special characters also prevents unwanted side effects such as exe-
 1348 cuting a command or writing a file.

1349 **History of Decisions Made**

1350 This function was added in Draft 9 as an alternative to *glob()*. There has been
 1351 continuing controversy over exactly what features should be included in *glob()*. It
 1352 is hoped that providing *wordexp()* (which provides all of the shell's word expan-
 1353 sions, but will probably be slow to execute), and *glob()* (which is faster but does
 1354 only expansion of pathnames, without tilde or parameter expansion), will satisfy
 1355 the majority of reviewers.

1356 While *wordexp()* could be implemented entirely as a library routine, it is expected 1
 1357 that most implementations will run a shell in a subprocess to do the expansion.

1358 Two different approaches have been proposed for how the required information
 1359 might be presented to the shell and the results returned. They are presented
 1360 here as examples.

1361 One proposal is to extend the *echo* utility by adding a *-q* option. This option
 1362 would cause *echo* to add a backslash before each backslash and each <blank>
 1363 that occurs within an argument. The *wordexp()* function could then invoke the
 1364 shell as follows:

```
1365     (void) strcpy (buffer, "echo -q ");
1366     (void) strcat (buffer, words);
1367     if ((flags & WRDE_SHOWERR) == 0)
1368         (void) strcat (buffer, " 2>/dev/null");
1369     f = popen (buffer, "r");
```

1370 The *wordexp()* function would read the resulting output, remove unquoted
 1371 backslashes, and break into words at unquoted <blank>s. If the WRDE_NOCMD
 1372 flag was set, *wordexp()* would have to scan *words* before starting the subshell to
 1373 make sure that there would be no command substitution. In any case, it would
 1374 have to scan *words* for unquoted special characters.

1375 Another proposal is to add the following options to *sh*:

1376 `-w wordlist` This option provides a wordlist expansion service to applications.
 1377 The words in *wordlist* are expanded, and the following is written
 1378 to standard output:

- 1379 (1) The count of the number of words after expansion, in
 1380 decimal, followed by a null byte.
- 1381 (2) The number of bytes needed to represent the expanded
 1382 words (not including null separators), in decimal, followed
 1383 by a null byte.
- 1384 (3) The expanded words, each terminated by a null byte.

1385 If an error is encountered during word expansion, `sh` exits with a
 1386 nonzero status after writing the above to report any words suc-
 1387 cessfully expanded

1388 `-P` Run in “protected” mode. If specified with the `-w` option, no com-
 1389 mand substitution is performed.

1390 With these options, *wordexp()* could be implemented fairly simply by creating a
 1391 subprocess using *fork()*, and executing `sh` using the line:

1392 `execl(<shell path>, "sh", "-P", "-w", words, (char *)0);`

1393 after directing standard error to `/dev/null`.

1394 It seemed objectionable for a library routine to write messages to standard error,
 1395 unless explicitly requested, so *wordexp()* is required to redirect standard error to
 1396 `/dev/null` to ensure that no messages are generated, even for commands exe-
 1397 cuted for command substitution. The new `WRDE_SHOWERR` flag can be specified
 1398 to request that error messages be written.

1399 The `WRDE_REUSE` flag allows the implementation to avoid the expense of freeing
 1400 and reallocating memory, if that is possible. A minimal implementation can just
 1401 call *wordfree()* when `WRDE_REUSE` is set.

B.10 C Binding for Get POSIX Configurable Variables

B.10.1 C Binding for Get String-Valued Configurable Variables

Function: *confstr()*

B.10.1.1 Synopsis

```
#include <unistd.h>
```

```
size_t confstr(int name, char *buf, size_t len);
```

B.10.1.2 Description

The *confstr()* function provides a method for applications to get configuration-defined string values. Its use and purpose are similar to the *sysconf()* function defined in POSIX.1 {8}, but it is used where string values rather than numeric values are returned.

The *name* argument represents the system variable to be queried. The implementation shall support all of the *name* values shown in Table B-18, which are defined in *<unistd.h>*. It may support others.

Table B-18 – *confstr()* *name* Values

<i>name</i> Value	String returned by <i>confstr()</i>
<i>_CS_PATH</i>	A value for the PATH environment variable that finds all standard utilities.

If *len* is not zero, and if *name* has a configuration-defined value, *confstr()* shall copy that value into the *len*-byte buffer pointed to by *buf*. If the string to be returned is longer than *len* bytes, including the terminating null, then *confstr()* shall truncate the string to *len*–1 bytes and null-terminate the result. The application can detect that the string was truncated by comparing the value returned by *confstr()* with *len*.

If *len* is zero and *buf* is **NULL**, then *confstr()* still shall return the integer value as defined below, but shall not return a string. If *len* is zero but *buf* is not **NULL**, the result is unspecified.

B.10.1.3 Returns

If *name* does not have a configuration-defined value, *confstr()* shall return zero and leave *errno* unchanged.

If *name* has a configuration-defined value, the *confstr()* function shall return the size of buffer that would be needed to hold the entire configuration-defined value.

1436 If this return value is greater than *len*, the string returned in *buf* has been
 1437 truncated.

1438 **B.10.1.4 Errors**

1439 If any of the following conditions occur, *confstr()* shall return zero and set *errno* to
 1440 the corresponding value:

1441 [EINVAL] The value of the *name* argument is invalid.

1442 **B.10.1.5 Rationale.** (*This subclause is not a part of P1003.2*)

1443 **Examples, Usage**

1444 An application can distinguish between an invalid *name* parameter value and one
 1445 that corresponds to a configurable variable that has no configuration-defined
 1446 value by checking if *errno* has been modified. This mirrors the behavior of *sys-*
 1447 *conf()* in POSIX.1 {8}.

1448 The original need for this function was to provide a way of finding the
 1449 configuration-defined default value for the environment variable **PATH**. Since
 1450 **PATH** can be modified by the user to include directories that could contain utili-
 1451 ties replacing POSIX.2 standard utilities, applications need a way to determine
 1452 the system-supplied **PATH** environment variable value that contains the correct
 1453 search path for the POSIX.2 standard utilities.

1454 An application could use *confstr(name, NULL, (size_t) 0)* to find out how big
 1455 a buffer is needed for the string value, *malloc()* a buffer to hold the string, and
 1456 call *confstr()* again to get the string. Alternately, it could allocate a fixed, static
 1457 buffer that is big enough to hold most answers (512 bytes, maybe, or 1024), but
 1458 then *malloc()* a larger buffer if it finds that this is too small.

1459 **History of Decisions Made**

1460 In Draft 7, these values and *sysconf()* values defined in POSIX.1 {8} were obtained
 1461 using a function named *posixconf()*. However, that routine was dropped in favor
 1462 of *csysconf()*. There did not seem to be any reason to provide the redundant inter-
 1463 face to POSIX.1 {8} functions, nor to return values as strings when numeric values
 1464 are really what are needed. *csysconf()* could be extended to return strings for
 1465 other related standards or features.

1466 In Draft 9, *csysconf()* has been replaced by *confstr()*. The name was changed
 1467 because too many people were confused by the name; they thought that the ‘c’
 1468 referred to the C language, rather than characters (as distinct from integers).
 1469 The *confstr()* function also copies the returned string into a buffer supplied by the
 1470 application instead of returning a pointer to a string. This allows a cleaner inter-
 1471 face in some implementations (lightweight processes were mentioned), and
 1472 resolves questions about when the application must copy the string returned.

B.10.2 C Binding for Get Numeric-Valued Configurable Variables

Functions: *sysconf()*, *pathconf()*, *fpathconf()*

A system that supports the C Language Bindings Option shall support the C language bindings defined in POSIX.1 {8} for the *sysconf()*, *pathconf()*, and *fpathconf()* functions. Of the *name* values defined in POSIX.1 {8}, only those that correspond to numeric-valued configuration values listed in Table 7-1, are required by POSIX.2. In addition, the *sysconf()* function shall support the *name* values in Table B-19, defined in `<unistd.h>`, to provide values for values in 2.13.1.

Table B-19 – C Bindings for Numeric-Valued Configurable Variables

Symbolic Limit	<i>name</i> Value
{BC_BASE_MAX}	_SC_BC_BASE_MAX
{BC_DIM_MAX}	_SC_BC_DIM_MAX
{BC_SCALE_MAX}	_SC_BC_SCALE_MAX
{BC_STRING_MAX}	_SC_BC_STRING_MAX
{COLL_WEIGHTS_MAX}	_SC_COLL_WEIGHTS_MAX
{EXPR_NEST_MAX}	_SC_EXPR_NEST_MAX
{LINE_MAX}	_SC_LINE_MAX
{RE_DUP_MAX}	_SC_RE_DUP_MAX
{POSIX2_VERSION}	_SC_2_VERSION
{POSIX2_C_DEV}	_SC_2_C_DEV
{POSIX2_FORT_DEV}	_SC_2_FORT_DEV
{POSIX2_FORT_RUN}	_SC_2_FORT_RUN
{POSIX2_LOCALEDEF}	_SC_2_LOCALEDEF
{POSIX2_SW_DEV}	_SC_2_SW_DEV

B.10.3 Rationale. *(This subclause is not a part of P1003.2)*

In Draft 9, the *name* values corresponding to the `_POSIX2_*` symbolic limits were changed to more closely follow the convention used in POSIX.1 {8}. In POSIX.1 {8}, for example, the *name* value for `{_POSIX_VERSION}` is `_SC_VERSION`. The POSIX.2 *name* value for `{_POSIX2_C_DEV}` (actually, it was `{_POSIX_C_DEV}` in Draft 8) was `_SC_POSIX_C_DEV`, and is now `_SC_2_C_DEV`.

If `sysconf(_SC_2_VERSION)` is not equal to the value of the `{_POSIX2_VERSION}` symbolic constant (see B.2.2), the utilities available via *system()* or *popen()* might not behave as described in this standard. This would mean that the application is not running in an environment that conforms to POSIX.2. Some applications might be able to deal with this, others might not. However, the interfaces defined in Annex B shall continue to operate as specified, even if `sysconf(_SC_2_VERSION)` reports that the utilities no longer perform as specified.

1514 **B.11 C Binding for Locale Control**

1515 The C binding to the services described in 7.9 shall be the *setlocale()* function
1516 defined in POSIX.1 {8} 8.1.2. In addition to the category values defined in
1517 POSIX.1 {8}, *setlocale()* shall also accept the value LC_MESSAGES, which shall be
1518 defined in `<locale.h>`.

1519 **B.11.1 C Binding for Locale Control Rationale.** *(This subclause is not a part of* 1520 *P1003.2)*

1521 The order in which the various locale categories are processed by *setlocale()* is not
1522 specified by POSIX.1 {8}, so the place for LC_MESSAGES in that order is also
1523 unspecified.

Annex C (normative)

FORTRAN Development and Runtime Utilities Options

This annex describes utilities used for the development of FORTRAN language applications, including compilation or translation of FORTRAN source code, and the execution of certain FORTRAN applications at runtime.

The utilities described in this annex may be provided by the conforming system; however, any system claiming conformance to the FORTRAN Development Utilities Option shall provide the `fort77` utility and any system claiming conformance to the FORTRAN Runtime Utilities Option shall provide the `asa` utility.

C.0.1 FORTRAN Development and Runtime Utilities Options Rationale.

(This subclause is not a part of P1003.2)

This clause is included in this standard as a temporary measure to accommodate existing FORTRAN developers. It is the intention of the POSIX.2 working group that this annex be moved from this standard to the emerging standard being developed by the POSIX.9 working group, which will specify FORTRAN-specific interfaces to the basic services provided by this standard and POSIX.1. The movement of this annex should occur in a later version of this standard.

See the rationale for `asa` for a description of the FORTRAN Runtime Utilities Option and why it was split off from the FORTRAN Development Utilities Option.

C.1 `asa` — Interpret carriage-control characters

This utility is optional. It shall be provided on systems that support the FORTRAN Runtime Utilities Option.

C.1.1 Synopsis

`asa` [*file* ...]

C.1.2 Description

The `asa` utility shall write its input files to standard output, mapping carriage-control characters from the text files to line-printer control sequences in an implementation-defined manner.

The first character of every line shall be removed from the input, and the following actions shall be performed:

If the character removed is:

<space>	The rest of the line shall be output without change.
0	A <newline> shall be output, then the rest of the input line.
1	One or more implementation-defined characters that causes an advance to the next page shall be output, followed by the rest of the input line.
+	The <newline> of the previous line shall be replaced with one or more implementation-defined characters that causes printing to return to column position 1, followed by the rest of the input line. If the + is the first character in the input, it shall have the same effect as <space>.

The action of the `asa` utility is unspecified upon encountering any character other than those listed above as the first character in a line.

C.1.3 Options

None.

C.1.4 Operands

file A pathname of a text file used for input. If no *file* operands are specified, the standard input shall be used.

C.1.5 External Influences

C.1.5.1 Standard Input

The standard input shall be used only if no *file* operands are specified. See Input Files.

C.1.5.2 Input Files

The input files shall be text files.

53 **C.1.5.3 Environment Variables**

54 The following environment variables shall affect the execution of `asa`:

55	LANG	This variable shall determine the locale to use for the
56		locale categories when both LC_ALL and the correspond-
57		ing environment variable (beginning with LC_) do not
58		specify a locale. See 2.6.
59	LC_ALL	This variable shall determine the locale to be used to over-
60		ride any values for locale categories specified by the set-
61		tings of LANG or any environment variables beginning
62		with LC_ .
63	LC_CTYPE	This variable shall determine the locale for the interpreta-
64		tion of sequences of bytes of text data as characters (e.g.,
65		single- versus multibyte characters in arguments and
66		input files).
67	LC_MESSAGES	This variable shall determine the language in which mes-
68		sages should be written.

69 **C.1.5.4 Asynchronous Events**

70 Default.

71 **C.1.6 External Effects**

72 **C.1.6.1 Standard Output**

73 The standard output shall be the text from the input file modified as described in
74 C.1.2.

75 **C.1.6.2 Standard Error**

76 None.

77 **C.1.6.3 Output Files**

78 None.

79 **C.1.7 Extended Description**

80 None.

81 C.1.8 Exit Status

82 The `asa` utility shall exit with one of the following values:

83 0 All input files were output successfully.

84 >0 An error occurred.

85 C.1.9 Consequences of Errors

86 Default.

87 C.1.10 Rationale. *(This subclause is not a part of P1003.2)*

88 Examples, Usage

89 The `asa` utility is needed to map “standard” FORTRAN 77 output into a form
90 acceptable to contemporary printers. Usually `asa` is used to pipe data to the `lp`
91 utility (see `lp` in 4.38.)

92 The following command:

```
93       asa file
```

94 permits the viewing of `file` (created by a program using FORTRAN-style carriage
95 control characters) on a terminal.

96 The following command:

```
97       a.out | asa | lp
```

98 formats the FORTRAN output of `a.out` and directs it to the printer.

99 History of Decisions Made

100 This utility is generally used only by FORTRAN programs. It was moved to this
101 annex in response to multiple ballot objections requesting its removal. The work-
102 ing group decided to retain `asa` to avoid breaking the existing large base of FOR-
103 TRAN applications that put carriage control characters in their output files. This
104 is a compromise position to achieve balloting acceptance: the overhead of main-
105 taining a separate option in POSIX.2 for just this one utility is seen to be small in
106 comparison to the benefit achieved for FORTRAN applications. Since it is a
107 separate option, there is no requirement that a system have a FORTRAN compiler
108 in order to run applications that need `asa`.

109 Historical implementations have used an ASCII <form-feed> character in
110 response to a '1', and an ASCII <carriage-return> in response to a '+'. It is
111 suggested that implementations treat characters other than '0', '1', and '+' as
112 <space> in the absence of any compelling reason to do otherwise. However, the
113 action is listed here as “unspecified,” permitting an implementation to provide
114 extensions to access fast multiple line slewing and channel seeking in a nonport-
115 able manner.

C.2 `fort77` — FORTRAN compiler

This utility is optional. It shall be provided on systems that support the FORTRAN Development Utilities Option.

C.2.1 Synopsis

```
fort77 [-c] [-g] [-L directory] ... [-O optlevel] [-o outfile] [-s] [-w]
      operand ...
```

C.2.2 Description

The `fort77` utility is the interface to the FORTRAN compilation system; it shall accept the full FORTRAN language defined by ISO 1539 {2}. The system conceptually consists of a compiler and link editor. The files referenced by *operands* are compiled and linked to produce an executable file. (It is unspecified whether the linking occurs entirely within the operation of `fort77`; some systems may produce objects that are not fully resolved until the file is executed.)

If the `-c` option is present, for all pathname operands of the form *file.f*, the files

```
$(basename pathname .f).o
```

shall be created or overwritten as the result of successful compilation. If the `-c` option is not specified, it is unspecified whether such `.o` files are created or deleted for the *file.f* operands.

If there are no options that prevent link editing (such as `-c`) and all operands compile and link without error, the resulting executable file shall be written into the file named by the `-o` option (if present) or to the file `a.out`. The executable file shall be created as specified in 2.9.1.4, except that the file permissions shall be set to

```
S_IRWXO | S_IRWXG | S_IRWXU
```

(see POSIX.1 {8} 5.6.1.2) and that the bits specified by the *umask* of the process shall be cleared.

C.2.3 Options

The `fort77` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that:

- The `-l` *library* operands have the format of options, but their position within a list of operands affects the order in which libraries are searched.
- The order of specifying the multiple `-L` options is significant.
- Conforming applications shall specify each option separately; that is, grouping option letters (e.g., `-cg`) need not be recognized by all implementations.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

The following options shall be supported by the implementation:

- `-c` Suppress the link-edit phase of the compilation, and do not remove any object files that are produced.
- `-g` Produce symbolic information in the object or executable files; the nature of this information is unspecified, and may be modified by implementation-defined interactions with other options.
- `-s` Produce object and/or executable files from which symbolic and other information not required for proper execution using the POSIX.1 {8} *exec* family has been removed (stripped). If both `-g` and `-s` options are present, the action taken is unspecified.
- `-o outfile` Use the pathname *outfile*, instead of the default `a.out`, for the executable file produced. If the `-o` option is present with `-c`, the result is unspecified.
- `-L directory` Change the algorithm of searching for the libraries named in `-l` operands to look in the directory named by the *directory* pathname before looking in the usual places. Directories named in `-L` options shall be searched in the specified order. Implementations shall support at least ten instances of this option in a single `fort77` command invocation. If a directory specified by a `-L` option contains a file named `libf.a`, the results are unspecified.
- `-O optlevel` Specify the level of code optimization. If the *optlevel* option-argument is the digit 0, all special code optimizations shall be disabled. If it is the digit 1, the nature of the optimization is unspecified. If the `-O` option is omitted, the nature of the system's default optimization is unspecified. It is unspecified whether code generated in the presence of the `-O 0` option is the same as that generated when `-O` is omitted. Other *optlevel* values may be supported.
- `-w` Suppress warnings.

Multiple instances of `-L` options can be specified.

C.2.4 Operands

An *operand* is either in the form of a pathname or the form `-l library`. At least one operand of the pathname form shall be specified. The following operands shall be supported by the implementation:

- file.f* The pathname of a FORTRAN source file to be compiled and optionally passed to the link editor. The file name operand shall be of this form if the `-c` option is used.
- file.a* A library of object files typically produced by `ar` (see 6.1), and passed directly to the link editor. Implementations may recognize implementation-defined suffixes other than `.a` as denoting

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

192 object file libraries.

193 *file.o* An object file produced by `fort77 -c`, and passed directly to the

194 link editor. Implementations may recognize implementation-

195 defined suffixes other than `.o` as denoting object files.

196 The processing of other files is implementation defined.

197 `-l library` (The letter ell.) Search the library named:

198 `liblibrary.a`

199 A library is searched when its name is encountered, so the place-

200 ment of a `-l` operand is significant. Several standard libraries

201 can be specified in this manner, as described in C.2.7. Implemen-

202 tations may recognize implementation-defined suffixes other than

203 `.a` as denoting libraries.

204 C.2.5 External Influences

205 C.2.5.1 Standard Input

206 None.

207 C.2.5.2 Input Files

208 The input file shall be one of the following: a text file containing FORTRAN source

209 code; an object file in the format produced by `fort77 -c`; or a library of object

210 files, in the format produced by archiving zero or more object files, using `ar`.

211 Implementations may supply additional utilities that produce files in these for-

212 mats. Additional input files are implementation defined.

213 A `<tab>` character encountered within the first six characters on a line of source

214 code shall cause the compiler to interpret the following character as if it were the

215 seventh character on the line (i.e., in column 7).

216 C.2.5.3 Environment Variables

217 The following environment variables shall affect the execution of `fort77`:

218 **LANG** This variable shall determine the locale to use for the

219 locale categories when both **LC_ALL** and the correspond-

220 ing environment variable (beginning with **LC_**) do not

221 specify a locale. See 2.6.

222 **LC_ALL** This variable shall determine the locale to be used to over-

223 ride any values for locale categories specified by the set-

224 tings of **LANG** or any environment variables beginning

225 with **LC_**.

226	LC_CTYPE	This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g.,
227		single- versus multibyte characters in arguments and
228		input files).
229		
230	LC_MESSAGES	This variable shall determine the language in which messages should be written.
231		
232	TMPDIR	This variable shall be interpreted as a pathname that should override the default directory for temporary files, if
233		any.
234		

235 **C.2.5.4 Asynchronous Events**

236 Default.

237 **C.2.6 External Effects**

238 **C.2.6.1 Standard Output**

239 None.

240 **C.2.6.2 Standard Error**

241 Used only for diagnostic messages. If more than one file operand ending in .f (or
242 possibly other unspecified suffixes) is given, for each such file:

243 "%s:\n", <file>

244 may be written to allow identification of the diagnostic message with the
245 appropriate input file.

246 This utility may produce warning messages about certain conditions that do not
247 warrant returning an error (nonzero) exit value.

248 **C.2.6.3 Output Files**

249 Object files, listing files, and/or executable files shall be produced in unspecified
250 formats.

251 **C.2.7 Extended Description**

252 **C.2.7.1 Standard Libraries**

253 The `fort77` utility shall recognize the following `-l` operand for the standard
254 library:

255 `-l f` This library contains all library functions referenced in ISO 1539
 256 {2}. An implementation shall not require this operand to be
 257 present to cause a search of this library.

258 In the absence of options that inhibit invocation of the link editor, such as `-c`, the
 259 `fort77` utility shall cause the equivalent of a `-l f` operand to be passed to the
 260 link editor as the last `-l` operand, causing it to be searched after all other object
 261 files and libraries are loaded.

262 It is unspecified whether the library `libf.a` exists as a regular file. The imple-
 263 mentation may accept as `-l` operands names of objects that do not exist as regu-
 264 lar files.

265 **C.2.7.2 External Symbols**

266 The FORTRAN compiler and link editor shall support the significance of external 1
 267 symbols up to a length of at least 31 bytes. The compiler may fold case (i.e., may 1
 268 ignore uppercase/lowercase distinctions between identifiers). The action taken 1
 269 upon encountering symbols exceeding the implementation-defined maximum sym-
 270 bol length is unspecified.

271 The compiler and link editor shall support a minimum of 511 external symbols
 272 per source or object file, and a minimum of 4095 external symbols total. A diag-
 273 nostic message is written to standard output if the implementation-defined limit
 274 is exceeded; other actions are unspecified.

275 **C.2.8 Exit Status**

276 The `fort77` utility shall exit with one of the following values:

277 0 Successful compilation or link edit.
 278 >0 An error occurred.

279 **C.2.9 Consequences of Errors**

280 When `fort77` encounters a compilation error, it shall write a diagnostic to stan-
 281 dard error and continue to compile other source code operands. It shall return a
 282 nonzero exit status, but it is implementation defined whether an object module is
 283 created. If the link edit is unsuccessful, a diagnostic message shall be written to
 284 standard error, and `fort77` shall exit with a nonzero status.

285 **C.2.10 Rationale.** *(This subclause is not a part of P1003.2)*

286 **Examples, Usage**

287 The following are examples of usage:

288 `fort77 -o foo xyz.f` Compiles `xyz.f` and creates the executable `foo`.

289 `fort77 -c xyz.f` Compiles `xyz.f` and creates the object file `xyz.o`.

290 `fort77 xyz.f` Compiles `xyz.f` and creates the executable `a.out`.

291 `fort77 xyz.f b.o` Compiles `xyz.f`, links it with `b.o`, and creates the
292 executable `a.out`.

293 **History of Decisions Made**

294 The file inclusion and symbol definition (`#define`) mechanisms used by the `c89`
295 utility were not included in POSIX.2—even though they are commonly
296 implemented—since there is no requirement that the FORTRAN compiler use the
297 C preprocessor.

298 The `-onetrip` option was not included in this specification, even though many
299 historical compilers support it, because it is a relic from FORTRAN-66; it is an
300 anachronism that should not be perpetuated.

301 Some implementations produce compilation listings. This aspect of FORTRAN has
302 been left unspecified because there was opposition within the balloting group to
303 the various methods proposed for implementing it: a `-V` option overlapped with
304 historical vendor practice and a naming convention of creating files with `.l`
305 suffixes collided with historical `lex` file naming practice.

306 There is no `-I` option in this version of POSIX.2 to specify a directory for file inclu-
307 sion. An `INCLUDE` directive has been a part of the FORTRAN-8X discussions, but
308 it is not clear whether it will be retained.

309 It is noted that many FORTRAN compilers produce an object module even when
310 compilation errors occur; during a subsequent compilation, the compiler may
311 patch the object module rather than recompiling all the code. Consequently, it is
312 left to the implementor whether or not an object file is created.

313 The name of this utility was changed to `fort77` in Draft 9 to parallel the renam-
314 ing of the C compiler. The name `f77` was not chosen to avoid collision with his-
315 torical implementations.

316 A reference to MIL-STD-1753 was removed from an earlier draft in response to a
317 request from the POSIX.9 working group. It was not the intention of this docu-
318 ment to require certification of the FORTRAN compiler and the forthcoming
319 POSIX.9 standard does not specify the military standard or any special prepro-
320 cessing requirements. Furthermore, use of that document would have been inap-
321 propriate for an international standard.

322 The specification of optimization has been subject to changes through early drafts.
323 At one time, `-O` and `-N` were Booleans: optimize and do not optimize (with an

unspecified default). Some historical practice lead this to be changed to:

 -O 0 No optimization.

 -O 1 Some level of optimization.

 -O n Other, unspecified levels of optimization.

It is not always clear whether “good code generation” is the same thing as optimization. Simple optimizations of local actions do not usually affect the semantics of a program. The -O 0 option has been included to accommodate the very fussy nature of scientific calculations in a highly optimized environment; compilers make errors. Some degree of optimization is expected, even if it is not documented here, and the ability to shut it off completely could be important when porting an application. An implementation may treat -O 0 as “do less than normal” if it wishes, but this is only meaningful if any of the operations it performs can affect the semantics of a program. It is highly dependent on the implementation whether doing less than normal makes sense. It is not the intent of this to ask for sloppy code generation, but rather to assure that any semantically visible optimization is suppressed.

The specification of standard library access is consistent with the C compiler specification. Implementations are not required to have `/usr/lib/libf.a`, as many historical implementations do, but if not they are required to recognize ‘f’ as a token.

External symbol size limits are in a normative subclause; portable applications need to know these limits. However, the minimum maximum symbol length should be taken as a constraint on a portable application, not on an implementation, and consequently the action taken for a symbol exceeding the limit is unspecified. The minimum size for the external symbol table was added for similar reasons.

The Consequences of Errors subclause clearly specifies the compiler’s behavior when compilation or link-edit error occur. The behavior of several historical implementations was examined, and the choice was made to be silent on the status of the executable, or `a.out`, file in the face of compiler or linker errors. If a linker writes the executable file, then links it on disk with `lseek()`s and `write()`s, the partially-linked executable can be left on disk and its execute bits turned off if the link edit fails. However, if the linker links the image in memory before writing the file to disk, it need not touch the executable file (if it already exists) because the link edit fails. Since both approaches are existing practice, a portable application shall rely on the exit status of `fort77`, rather than on the existence or mode of the executable file.

The -g and -s options are not specified as mutually exclusive. Historically these two options have been mutually exclusive, but because both are so loosely specified, it seemed cleaner to leave their interaction unspecified.

The requirement that portable applications specify compiler options separately is to reserve the multicharacter option namespace for vendor-specific compiler options, which are known to exist in many historical implementations. Implementations are not required to recognize, for example, -gc as if it were -g -c; nor

368 are they forbidden from doing so. The synopsis shows all of the options separately
369 to highlight this requirement on applications.

370 Echoing filenames to standard error is considered a diagnostic message, because
371 it would otherwise difficult to associate an error message with the erring file.
372 They are describing with “may” to allow implementations to use other methods of
373 identifying files and to parallel the description in c89.

Annex D (informative)

Bibliography

- 1 {B1} ISO 639: 1988, *Code for the representation of names of languages*.¹⁾
- 2 {B2} ISO 2022: 1986, *Information processing—ISO 7-bit and 8-bit coded charac-*
- 3 *ter sets—Code extension techniques*.
- 4 {B3} ISO 2047: 1975, *Information processing—Graphical representations for the*
- 5 *control characters of the 7-bit coded character set*.
- 6 {B4} ISO 3166: 1988, *Code for the representation of names of countries*.
- 7 {B5} ISO 6429: 1988, *Information processing—Control functions for 7-bit and 8-*
- 8 *bit coded character sets*.
- 9 {B6} ISO 6937-2: 1983, *Information processing—Coded character sets for text*
- 10 *communication—Part 2: Latin alphabetic and non-alphabetic graphic*
- 11 *characters*.
- 12 {B7} ISO 8802-3: 1989, *Information processing systems—Local area networks—*
- 13 *Part 3: Carrier sense multiple access with collision detection (CSMA/CD)*
- 14 *access method and physical layer specification*.
- 15 {B8} ISO 8806: 1988, *Data elements and interchange formats—Information inter-*
- 16 *change —Representation of dates and times*.
- 17 {B9} ISO 8859, *Information processing—8-bit single-byte coded graphic character*
- 18 *sets*. (Parts 1 to 8 published.)
- 19 {B10} ISO/IEC 10367: ...,²⁾ *Information processing—Repertoire of standardized*
- 20 *coded graphic character sets for use in 8-bit codes*.
- 21 {B11} ISO/IEC 10646: ...,³⁾ *Information technology—Universal Coded Character*
- 22 *Set (UCS)*.

23 1) ISO documents can be obtained from the ISO office, 1, rue de Varembe, Case Postale 56, CH-1211,
24 Genève 20, Switzerland/Suisse.

25 2) To be approved and published.

26 3) To be approved and published.

- 27 {B12} International Organization for Standardization/Association Française de
28 Normalisation. *Dictionary of Computer Science/Dictionnaire de*
29 *L'Informatique*. Geneva/Paris: ISO/AFNOR, 1989.
- 30 {B13} ANSI X3.43-1986,⁴⁾ *Representations for Local Times of the Day for Informa-*
31 *tion Interchange*.
- 32 {B14} GB 2312-1980, Chinese Association for Standardization. *Coded Chinese*
33 *Graphic Character Set for Information Interchange*.
- 34 {B15} JIS X0208-1990, Japanese National Committee on ISO/IEC JTC1/SC2.
35 *Japanese Graphic Character Set for Information Interchange*.
- 36 {B16} JIS X0212-1990, Japanese National Committee on ISO/IEC JTC1/SC2. *Sup-*
37 *plementary Japanese Graphic Character Set for Information Interchange*.
- 38 {B17} KS C 5601-1987, Korean Bureau of Standards. *Korean Graphic Character*
39 *Set for Information Interchange*.
- 40 {B18} IEEE Std 100-1988, *IEEE Standard Dictionary of Electrical and Electronics*
41 *Terms*.
- 42 {B19} IEEE P1003.3,⁵⁾ *Standard for Information Technology—Test Methods for*
43 *Measuring Conformance to POSIX*
- 44 {B20} IEEE P1003.3.2,⁶⁾ *Standard for Information Technology—Test Methods for*
45 *Measuring Conformance to POSIX.2*
- 46 {B21} Aho, Alfred V., Kernighan, Brian W., Weinberger, Peter J., *The AWK Pro-*
47 *gramming Language*, Reading, MA: Addison-Wesley, 1988.
- 48 {B22} Aho, Alfred V., Sethi, Ravi, Ullman, Jeffrey D., *Compilers, Principles,*
49 *Techniques, and Tools*, Reading, MA: Addison-Wesley, 1986.
- 50 {B23} Aho, Alfred V., Ullman, Jeffrey D., *Principles of Compiler Design*, Reading,
51 MA: Addison-Wesley, 1977.
- 52 {B24} American Telephone and Telegraph Company. *System V Interface*
53 *Definition (SVID), Issues 2 and 3*. Morristown, NJ: UNIX Press, 1986,
54 1989.⁷⁾
- 55 {B25} Bolsky, Morris I., Korn, David G., *The KornShell Command and Program-*
56 *ming Language*, Englewood Cliffs, NJ: Prentice Hall, 1988.

57 4) ANSI documents can be obtained from the Sales Department, American National Standards
58 Institute, 1430 Broadway, New York, NY 10018.

59 5) To be approved and published.

60 6) To be approved and published.

61 7) This is one of several documents that represent an industry specification in an area related to
62 POSIX.2. The creators of such documents may be able to identify newer versions that may be
63 interesting.

- 64 {B26} DeRemer, Frank, and Thomas J. Pennello, "Efficient Computation of
65 LALR(1) Look-ahead Sets." *SigPlan Notices* 15:8, 176-187, August, 1979.
- 66 {B27} Knuth, D. E. "On the translation of languages from left to right." *Informa-
67 tion and Control* 8:6, 607-639.
- 68 {B28} University of California at Berkeley—Computer Science Research Group.
69 *4.3 Berkeley Software Distribution, Virtual VAX-11 Version*. Berkeley, CA:
70 The Regents of the University of California, April 1986.
- 71 {B29} /usr/group Standards Committee. *1984 /usr/group Standard*. Santa
72 Clara, CA: UniForum, 1984.
- 73 {B30} X/Open Company, Ltd. *X/Open Portability Guide, Issue 2*. Amsterdam:
74 Elsevier Science Publishers, 1987.
- 75 {B31} X/Open Company, Ltd. *X/Open Portability Guide, Issue 3*. Englewood
76 Cliffs, NJ: Prentice-Hall, 1989.

Annex E (informative)

Rationale and Notes

1 This annex summarizes the deliberations of the IEEE P1003.2 Working Group,
2 the committee charged by the IEEE Computer Society's Technical Committee on
3 Operating Systems and Operational Environments with devising an interface
4 standard for a shell and related utilities to support and extend POSIX.1.

5 The annex is being published along with the standard to assist in the process of
6 review. It contains historical information concerning the contents of the standard
7 and why features were included or discarded by the Working Group. It also con-
8 tains notes of interest to application programmers on recommended programming
9 practices, emphasizing the consequences of some aspects of the standard that may
10 not be immediately apparent.

11 Just as this standard relies on the knowledge of architecture, history, and
12 definitions from the POSIX.1, so does this annex. The reader is referred to the
13 Rationale and Notes appendix of POSIX.1 for background material and biblio-
14 graphic information about UNIX systems in general and POSIX specifically, which
15 will not be duplicated here.

16 **E.1 General**

17 *Editor's Note: The text of the Rationale for this section has been temporarily*
18 *located in Section 1, adjacent to the text it is explaining. The text will return to*
19 *this annex after the completion of balloting.*

20 **E.1.1 Scope**

21 **E.1.2 Normative References**

22 **E.1.3 Conformance**

23 **E.2 Terminology and General Requirements**

24 *Editor's Note: The text of the Rationale for this section has been temporarily*
25 *located in Section 2, adjacent to the text it is explaining. The text will return to*
26 *this annex after the completion of balloting.*

27 **E.2.1 Conventions**

28 **E.2.2 Definitions**

29 **E.2.3 Built-in Utilities**

30 **E.2.4 Character Set**

31 **E.2.5 Locale**

32 **E.2.6 Environment Variables**

33 **E.2.7 Required Files**

34 **E.2.8 Regular Expression Notation**

35 **E.2.9 Dependencies on Other Standards**

36 **E.2.10 Utility Conventions**

37 **E.2.11 Utility Description Defaults**

38 **E.2.12 File Format Notation**

E.2.13 Configuration Values

E.3 Shell Command Language

Editor's Note: The text of the Rationale for this section has been temporarily located in Section 3, adjacent to the text it is explaining. The text will return to this annex after the completion of balloting.

E.3.1 Shell Definitions

E.3.2 Quoting

E.3.3 Token Recognition

E.3.4 Reserved Words

E.3.5 Parameters and Variables

E.3.6 Word Expansions

E.3.7 Redirection

E.3.8 Exit Status for Commands

E.3.9 Shell Commands

E.3.10 Shell Grammar

E.3.11 Signals and Error Handling

E.3.12 Shell Execution Environment

E.3.13 Pattern Matching Notation

E.3.14 Special Built-in Utilities

E.4 Execution Environment Utilities

Editor's Note: The text of the Rationale for this section has been temporarily located in Section 4, adjacent to the text it is explaining. The text will return to this annex after the completion of balloting. Notations regarding utilities probably included in the UPE have been updated, without diff marks, based on the current working draft of 1003.2a.

Many utilities were evaluated by the working group; more utilities were excluded from the standard than included. The following list contains many common UNIX system utilities that were not included as Execution Environment Utilities or in one of the Software Development Environment groups. It is logistically difficult for this Rationale to correctly distribute the reasons for not including a utility among the various utility environment sections. Therefore, this section covers the reasons for all utilities not included in Sections 4 and 6 and Annexes A and C.

The working group started its deliberations with a recommended list of utilities provided by the X/Open group of companies. This list was a subset of the utilities in the *X/Open Portability Guide, Issue II*, so it was very closely related to System V. The list had already been purged of purely administrative utilities, such as those found in System V's Administered System Extension. Then, the working group applied its scope as a filter and substantially pruned the remaining list as well.

The following list of "rejected" utilities is limited by its historical roots; since the selected utilities emerged from primarily a System V base, this list does not include sometimes familiar entries from BSD. The working group received substantial input from representatives of the University of California at Berkeley and from companies that are firmly allied with BSD versions of the UNIX system, enough so that some BSD-derived utilities are included in the standard. However, this Rationale is now limited to a discussion of only those utilities actively or indirectly evaluated by the working group, rather than the list of all known UNIX utilities from all its variants. This list will most likely be augmented during the balloting process as balloters request specific rationales for their favorite commands.

In the list, the notation *[POSIX.2a]* is used to identify utilities that are being evaluated for inclusion in the forthcoming User Portability Extension to this standard. Similarly, *[POSIX.7]* is used for those that may be appropriate for the working group evaluating system administration and *[POSIX.Net]* for networking standards.

94 adb 95 96	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging
--------------------------	--

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

97		tool. Furthermore, many useful aspects of <code>adb</code> are very
98		hardware-specific.
99	<code>admin</code>	The intent of the various software development utilities was to
100		assist in the installation (rather than the actual development and
101		debugging) of applications. This SCCS utility is primarily a
102		development tool.
103	<code>as</code>	Assemblers are hardware-specific and are included implicitly as
104		part of the compilers in the standard.
105	<code>at</code>	The <code>at</code> and <code>cron</code> family of utilities were omitted because portable
106		applications could not rely on their behavior. [POSIX.2a]
107	<code>banner</code>	The only known use of this command is as part of the LP printer
108		header pages. It was decided that the format of the header is
109		implementation defined, so this utility is superfluous to applica-
110		tion portability.
111	<code>batch</code>	The <code>at</code> and <code>cron</code> family of utilities were omitted because portable
112		applications could not rely on their behavior. [POSIX.2a]
113	<code>cal</code>	This calendar printing program is not useful to portable applica-
114		tions.
115	<code>calendar</code>	This reminder service program is not useful to portable applica-
116		tions.
117	<code>cancel</code>	The LP (line printer spooling) system specified is the most basic
118		possible and did not need this level of application control.
119		[POSIX.7]
120	<code>cflow</code>	The intent of the various software development utilities was to
121		assist in the installation (rather than the actual development and
122		debugging) of applications. This utility is primarily a debugging
123		tool.
124	<code>chroot</code>	This is primarily of administrative use, requiring super-user
125		privileges. [POSIX.7]
126	<code>col</code>	No utilities defined in this standard produce output requiring
127		such a filter. The <code>nroff</code> text formatter is present on many his-
128		torical systems and will continue to remain as an extension; <code>col</code>
129		is expected to be shipped by all the systems that ship <code>nroff</code> .
130	<code>cpio</code>	This has been replaced by <code>pax</code> , for reasons explained in its own
131		Rationale.
132	<code>c++</code>	Can be subsumed by <code>c89</code> .
133	<code>crontab</code>	The <code>at</code> and <code>cron</code> family of utilities were omitted because portable
134		applications could not rely on their behavior. [POSIX.2a]
135	<code>csplit</code>	This utility's functionality can sometimes be provided by the <code>dd</code>
136		or <code>sed</code> utilities (i.e., although these utilities cannot easily provide
137		all of <code>csplit</code> 's features in one package, they can frequently be

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

138		used for the type of task that <code>csplit</code> is being used for).
139		[<i>POSIX.2a</i>]
140	<code>cu</code>	Terminal oriented—not useful from shell scripts or typical application programs. [<i>POSIX.Net</i>]
141		
142	<code>cxref</code>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This utility is primarily a debugging tool.
143		
144		
145		
146	<code>dc</code>	This utility's functionality can be provided by the <code>bc</code> utility; <code>bc</code> was selected because it was easier to use and had superior functionality. Although the historical versions of <code>bc</code> are implemented using <code>dc</code> as a base, this standard prescribes the interface and not the underlying mechanism used to implement it.
147		
148		
149		
150		
151	<code>delta</code>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This SCCS utility is primarily a development tool.
152		
153		
154		
155	<code>df</code>	As the standard does not address the concept or nature of file systems, this command could not be specified in a manner useful to portable applications. [<i>POSIX.2a</i>]
156		
157		
158	<code>dircmp</code>	Although a useful concept, the traditional output of this directory comparison program is not suitable for processing in applications programs. Also, the <code>diff -r</code> command gives equivalent functionality.
159		
160		
161		
162	<code>dis</code>	Disassemblers are hardware-specific.
163	<code>du</code>	Because of differences between systems in measuring disk usage, this utility could not be used reliably by a portable application. [<i>POSIX.2a</i>]
164		
165		
166	<code>egrep</code>	Marked obsolescent and replaced by the new version of <code>grep</code> .
167	<code>ex</code>	This is typically a link to the <code>vi</code> terminal-oriented editor—not useful from shell scripts or typical application programs. The nonterminal oriented facilities of <code>ex</code> are provided by <code>ed</code> . [<i>POSIX.2a</i>]
168		
169		
170		
171	<code>fgrep</code>	Marked obsolescent and replaced by the new version of <code>grep</code> .
172	<code>file</code>	Determining the type of file is generally accomplished with <code>test</code> or <code>find</code> . The added information available with <code>file</code> is of little use to a portable application, particularly since there is considerable variation in its output contents. [<i>POSIX.2a</i>]
173		
174		
175		
176	<code>get</code>	The intent of the various software development utilities was to assist in the installation (rather than the actual development and debugging) of applications. This SCCS utility is primarily a development tool.
177		
178		
179		

180	<code>ld</code>	Is subsumed by <code>c89</code> .
181	<code>line</code>	The functionality of <code>line</code> can be provided with <code>read</code> .
182	<code>lint</code>	The intent of the various software development utilities was to
183		assist in the installation (rather than the actual development and
184		debugging) of applications. This utility is primarily a debugging
185		tool.
186	<code>login</code>	Terminal oriented—not useful from shell scripts or typical appli-
187		cation programs.
188	<code>lorder</code>	This utility is an aid in creating an implementation-specific detail
189		of object libraries that the working group did not feel required
190		standardization.
191	<code>lpstat</code>	The LP system specified is the most basic possible and did not
192		need this level of application control. [POSIX.7]
193	<code>m4</code>	The working group did not find that this macro processor had
194		sufficiently wide usage for standardization.
195	<code>mail</code>	This utility was omitted in favor of <code>mailx</code> , because there was a
196		considerable functionality overlap between the two. The mail-
197		sending aspects of <code>mailx</code> are covered in this standard, the mail-
198		reading in the UPE. [POSIX.2a]
199	<code>mesg</code>	Terminal oriented—not useful from shell scripts or typical appli-
200		cation programs. [POSIX.2a]
201	<code>mknod</code>	This was omitted in favor of <code>mkfifo</code> , as <code>mknod</code> has too many
202		implementation-defined functions. [POSIX.7]
203	<code>newgrp</code>	Terminal oriented—not useful from shell scripts or typical appli-
204		cation programs. [POSIX.2a]
205	<code>news</code>	Terminal oriented—not useful from shell scripts or typical appli-
206		cation programs.
207	<code>nice</code>	Due to historical variations in usage, and in the lack of underly-
208		ing support from possible POSIX.1 {8} base systems, this cannot be
209		used by applications to achieve reliable results. [POSIX.2a]
210	<code>nl</code>	The useful functionality of <code>nl</code> can be provided with <code>pr</code> .
211	<code>nm</code>	The intent of the various software development utilities was to
212		assist in the installation (rather than the actual development and
213		debugging) of applications. This utility is primarily a debugging
214		tool. [POSIX.2a]
215	<code>pack</code>	The working group found little interest in a portable data
216		compression program (and there are others that are probably
217		more widely used anyway).
218	<code>passwd</code>	Terminal oriented—not useful from shell scripts or typical appli-
219		cation programs. (There was also sentiment to avoid security-

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

220		related utilities until requirements of 1003.6 are known.)	
221	pcat	The working group found little interest in a portable data	
222		compression program (and there are others that are probably	
223		more widely used anyway).	
224	pg	Terminal oriented—not useful from shell scripts or typical appli-	
225		cation programs.	
226	prof	The intent of the various software development utilities was to	
227		assist in the installation (rather than the actual development and	
228		debugging) of applications. This utility is primarily a debugging	
229		tool.	
230	prs	The intent of the various software development utilities was to	
231		assist in the installation (rather than the actual development and	
232		debugging) of applications. This SCCS utility is primarily a	
233		development tool.	
234	ps	This utility has historically been difficult to specify portably due	
235		to the many implementation-defined aspects of processes. Furth-	
236		ermore, a portable application can rarely rely on information	
237		about what other processes are doing, as security mechanisms	
238		may prevent it. A process requiring one of its children's process	
239		IDs (such as for use with the <code>kill</code> command) will have to record	
240		the IDs at the time of creation. [POSIX.2a]	
241	red	Restricted editor. This was not considered by the working group	
242		because it never provided the level of security restriction	
243		required.	
244	rmdel	The intent of the various software development utilities was to	
245		assist in the installation (rather than the actual development and	
246		debugging) of applications. This SCCS utility is primarily a	
247		development tool.	
248	rsh	Restricted shell. This was not considered by the working group	
249		because it does not provide the level of security restriction that is	1
250		implied by historical documentation.	1
251	sact	The intent of the various software development utilities was to	
252		assist in the installation (rather than the actual development and	
253		debugging) of applications. This SCCS utility is primarily a	
254		development tool.	
255	sdb	The intent of the various software development utilities was to	
256		assist in the installation (rather than the actual development and	
257		debugging) of applications. This utility is primarily a debugging	
258		tool. Furthermore, some useful aspects of <code>sdb</code> are very	
259		hardware-specific.	
260	sdiff	The “side-by-side <code>diff</code> ” utility from System V was omitted	
261		because it is used infrequently, and even less so by portable appli-	
262		cations. Despite being in System V, it is not in the <i>SVID</i> or <i>XPG</i> .	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

263	shar	Utilities with this type of functionality (“shell-based archivers”)
264		are in wide use, despite not being included in System V or BSD
265		systems. However, the working group felt this sort of program
266		was more widely used by human users than portable applications.
267	shl	Terminal oriented—not useful from shell scripts or typical appli-
268		cation programs. The job control aspects of the Shell Command
269		Language are generally more useful and are being evaluated for
270		the UPE.
271	size	The intent of the various software development utilities was to
272		assist in the installation (rather than the actual development and
273		debugging) of applications. This utility is primarily a debugging
274		tool.
275	spell	Not useful from shell scripts or typical application programs.
276	split	The functionality can sometimes be provided by the dd, sed, or
277		(for some uses) xargs utilities (i.e., although these utilities can-
278		not easily provide all of split’s features in one package, they can
279		sometimes be used for the type of task that split is being used
280		for). [POSIX.2a]
281	strings	This is normally used by human users during debugging, rather
282		than by applications. [POSIX.2a]
283	su	Not useful from shell scripts or typical application programs.
284		(There was also sentiment to avoid security-related utilities until
285		requirements of POSIX.6 are known.)
286	sum	This utility was renamed cksum.
287	tabs	Terminal oriented—not useful from shell scripts or typical appli-
288		cation programs. [POSIX.2a]
289	time	Not necessary for portable applications. It is frequently used by
290		human users in debugging or for informal benchmarks. It is
291		doubtful whether any standardized definitions of the output could
292		be agreed upon.
293	tsort	This utility is an aid in creating an implementation-specific detail
294		of object libraries that the working group did not feel required
295		standardization.
296	unget	The intent of the various software development utilities was to
297		assist in the installation (rather than the actual development and
298		debugging) of applications. This SCCS utility is primarily a
299		development tool.
300	unpack	The working group found little interest in a portable data
301		compression program (and there are others that are probably
302		more widely used anyway).

303	uucp		
304	uulog		
305	uupick		
306	uustat		
307	uuto	The UUCP utilities and their protocol description were removed	1
308		from an early draft because responsibility for them was officially	1
309		requested by the POSIX group developing networking interfaces.	1
310	val	The intent of the various software development utilities was to	
311		assist in the installation (rather than the actual development and	
312		debugging) of applications. This SCCS utility is primarily a	
313		development tool.	
314	vi	Terminal oriented—not useful from shell scripts or typical appli-	
315		cation programs. [POSIX.2a]	
316	wall	Terminal oriented—not useful from shell scripts or typical appli-	
317		cation programs. It is generally used by system administrators,	
318		as well. [POSIX.7]	
319	what	The intent of the various software development utilities was to	
320		assist in the installation (rather than the actual development and	
321		debugging) of applications. This SCCS utility is primarily a	
322		development tool.	
323	who	The ability to determine other users on the system was felt to be	
324		at risk in a trusted implementation, so its use could not be con-	
325		sidered by a portable application. [POSIX.2a]	
326	write	Terminal oriented—not useful from shell scripts or typical appli-	
327		cation programs. [POSIX.2a]	

328 **E.4.1 awk — Pattern scanning and processing language**

329 **E.4.2 basename — Return nondirectory portion of pathname**

330 **E.4.3 bc — Arbitrary-precision arithmetic language**

331 **E.4.4 cat — Concatenate and print files**

332 **E.4.5 cd — Change working directory**

- 333 **E.4.6 chgrp — Change file group ownership**
- 334 **E.4.7 chmod — Change file modes**
- 335 **E.4.8 chown — Change file ownership**
- 336 **E.4.9 cksum — Write file checksums and block counts**
- 337 **E.4.10 cmp — Compare two files**
- 338 **E.4.11 comm — Select or reject lines common to two files**
- 339 **E.4.12 command — Select or reject lines common to two files**
- 340 **E.4.13 cp — Copy files**
- 341 **E.4.14 cut — Cut out selected fields of each line of a file**
- 342 **E.4.15 date — Write the date and time**
- 343 **E.4.16 dd — Convert and copy a file**
- 344 **E.4.17 diff — Compare two files**
- 345 **E.4.18 dirname — Return directory portion of pathname**
- 346 **E.4.19 echo — Write arguments to standard output**
- 347 **E.4.20 ed — Edit text**
- 348 **E.4.21 env — Set environment for command invocation**

349 **E.4.22** `expr` — Evaluate arguments as an expression

350 **E.4.23** `false` — Return false value

351 **E.4.24** `find` — Find files

352 **E.4.25** `fold` — Filter for folding lines

353 **E.4.26** `getconf` — Get configuration values

354 **E.4.27** `getopts` — Parse utility options

355 **E.4.28** `grep` — File pattern searcher

356 **E.4.29** `head` — Copy the first part of files

357 **E.4.30** `id` — Return user identity

358 **E.4.31** `join` — Relational database operator

359 **E.4.32** `kill` — Terminate or signal processes

360 **E.4.33** `ln` — Link files

361 **E.4.34** `locale` — Get locale-specific information

362 **E.4.35** `localedef` — Define locale environment

363 **E.4.36** `logger` — Log messages

364 **E.4.37** `logname` — Return user's login name

- 365 **E.4.38** `lp` — Send files to a printer
- 366 **E.4.39** `ls` — List directory contents
- 367 **E.4.40** `mailx` — Process messages
- 368 **E.4.41** `mkdir` — Make directories
- 369 **E.4.42** `mkfifo` — Make FIFO special files
- 370 **E.4.43** `mv` — Move files
- 371 **E.4.44** `nohup` — Invoke a utility immune to hangups
- 372 **E.4.45** `od` — Dump files in various formats
- 373 **E.4.46** `paste` — Merge corresponding or subsequent lines of files
- 374 **E.4.47** `pathchk` — Check pathnames
- 375 **E.4.48** `pax` — Portable archive interchange
- 376 **E.4.49** `pr` — Print files
- 377 **E.4.50** `printf` — Write formatted output
- 378 **E.4.51** `pwd` — Return working directory name
- 379 **E.4.52** `read` — Read a line from standard input
- 380 **E.4.53** `rm` — Remove directory entries

- 381 **E.4.54 rmdir — Remove directories**
- 382 **E.4.55 sed — Stream editor**
- 383 **E.4.56 sh — Shell, the standard command language interpreter**
- 384 **E.4.57 sleep — Suspend execution for an interval**
- 385 **E.4.58 sort — Sort, merge, or sequence check text files**
- 386 **E.4.59 stty — Set the options for a terminal**
- 387 **E.4.60 tail — Copy the last part of a file**
- 388 **E.4.61 tee — Duplicate standard input**
- 389 **E.4.62 test — Evaluate expression**
- 390 **E.4.63 touch — Change file access and modification times**
- 391 **E.4.64 tr — Translate characters**
- 392 **E.4.65 true — Return true value**
- 393 **E.4.66 tty — Return user's terminal name**
- 394 **E.4.67 umask — Get or set the file mode creation mask**
- 395 **E.4.68 uname — Return system name**
- 396 **E.4.69 uniq — Report or filter out repeated lines in a file**

E.4.70 wait — Await process completion**E.4.71 wc — Word, line, and byte count****E.4.72 xargs — Construct argument list(s) and invoke utility****E.5 User Portability Utilities Option**

Editor's Note: This section is unused in this revision of the standard.

E.6 Software Development Utilities Option

Editor's Note: The text of the Rationale for this section has been temporarily located in Section 6, adjacent to the text it is explaining. The text will return to this annex after the completion of balloting.

This is the first of the optional utility environments. The working group decided there were two basic classes of systems to be supported: general application execution and software development. The first is widely used and is the primary reason for the development of this standard. The second, however, represents only a (small?) subset of the first; the users are generally only those who are developing or installing C or FORTRAN applications.

Therefore, all the development environments are optional, giving users the option of specifying a smaller, (presumably) less expensive system. There are three separate optional environments, so that C-only or FORTRAN-only users do not have to specify unneeded components. As further languages are supported by this standard, their environments will also be optional.

An implementation must provide all three of these utilities to claim conformance to this section.

See section **E.4** for a discussion of utilities excluded from this group.

E.6.1 ar — Create and maintain library archives**E.6.2 make — Maintain, update, and regenerate groups of programs**

E.6.3 strip — Remove unnecessary information from executable files

E.7 Language-Independent System Services

Editor's Note: The text of the Rationale for this section has been temporarily located in Section 7, adjacent to the text it is explaining. The text will return to this annex after the completion of balloting.

E.7.1 Shell Command Interface

E.7.2 Access Environment Variables

E.7.3 Regular Expression Matching

E.7.4 Pattern Matching

E.7.5 Command Option Parsing

E.7.6 Generate Pathnames Matching a Pattern

E.7.7 Perform Word Expansions

E.7.8 Get POSIX Configurable Variables

E.7.9 Locale Control

E.8 C Language Development Utilities Option

Editor's Note: The text of the Rationale for this section has been temporarily located in Annex A, adjacent to the text it is explaining. The text will return to this annex after the completion of balloting.

This is the second of the optional utility environments.

An implementation must provide all three of these utilities to claim conformance to this section.

See section **E.4** for a discussion of utilities excluded from this group.

E.8.1 `c89` — Compile Standard C programs**E.8.2 `lex` — Generate programs for lexical tasks****E.8.3 `yacc` — Yet another compiler compiler****E.9 C Language Bindings Option**

Editor's Note: The text of the Rationale for this section has been temporarily located in Annex B, adjacent to the text it is explaining. The text will return to this annex after the completion of balloting.

E.9.1 C Language Definitions**E.9.2 C Numerical Limits****E.9.3 C Binding for Shell Command Interface****E.9.4 C Binding for Access Environment Variables****E.9.5 C Binding for Regular Expression Matching****E.9.6 C Binding for Match Filename or Pathname****E.9.7 C Binding for Command Option Parsing****E.9.8 C Binding for Generate Pathnames Matching a Pattern****E.9.9 C Binding for Perform Word Expansions****E.9.10 C Binding for Get POSIX Configurable Variables**

461 **E.9.11 C Binding for Locale Control**

462 **E.10 FORTRAN Development and Runtime Utilities Options**

463 *Editor's Note: The text of the Rationale for this section has been temporarily*
464 *located in Annex C, adjacent to the text it is explaining. The text will return to this*
465 *annex after the completion of balloting.*

466 This is the third and fourth of the optional utility environments.

467 See section **E.4** for a discussion of utilities excluded from this group.

468 **E.10.1 asa — Interpret carriage control characters**

469 **E.10.2 fort77 — FORTRAN compiler**

Annex F (informative)

Sample National Profile

Editor's Note: All uses of the term "character set" this annex have been changed to "coded character set" without further diff marks.

This annex is an example of a country's needs with respect to this standard and how those needs relate to other international standards as well as national standards. The example provided is included here for informative purposes and is not a formal standard in the country in question. It is provided by the Danish Standards Association¹⁾ and is as accurate as possible with regards to Danish needs.

F.1 (Example) Danish National Profile

This is the definition of the Danish Standards Association POSIX.2 profile. The subset of conforming implementations that provide the required characteristics below is referred to as conforming to the "Danish Standards Association (DS) Environment Profile" for this standard.

This profile specifies the following requirements on implementations:

- (1) In POSIX.2 section 2.13.1, the limit {COLL_WEIGHTS_MAX} shall be provided with a value of 4. All other limits shall conform to at least the minimum values shown in Table 2-16.
- (2) The following options shall be supported according to POSIX.2 section 2.13.2:

1) Further information may be obtained from the Danish Standards Association, Attn: S142u22A8 Baunegaardsvej 73, DK-2900 Hellerup, Denmark; FAX: +45 39 77 02 02; Email: u22a8@dkuug.dk

The data is also available electronically by anonymous FTP or FTAM at the site dkuug.dk in the directory i18n, where some other example national profiles, locales, and *charmaps* may also be found. They are also available by an archive server reached at archive@dkuug.dk; use "Subject: help" for further information.

26	POSIX2_C_BIND	Optional.	2
27	POSIX2_C_DEV	Optional.	2
28	POSIX2_FORT_DEV	Optional.	2
29	POSIX2_FORT_RUN	Optional.	2
30	POSIX2_LOCALEDEF	Required; the system shall support the creation of locales as described in 4.35.	2
31			2
32	POSIX2_SW_DEV	Optional.	2

F.1.1 Danish Locale Model

Editor's Note: This subclause is offered as rationale for the current state of this example annex. It will not necessarily appear in this form in any final version of the annex.

Creating a national locale for Denmark has been a quite elaborate effort. Time and again, we thought we had reached an agreement on the locale, but then some aspect disrupted the entire work, and we more or less had to start all over.

We think we have identified the cause of these problems to a general uncertainty regarding the exact purpose of a “national” locale. If we look at the Danish situation (which we know pretty well by now), we have identified several levels of locales, depending on the “complexity” of the collating sequence (or more generally sorting different kinds of text):

- (1) *Byte/machine level.* Here everything is sorted according to the character's byte value.
- (2) *Character/utility level.* Here we want to work almost on the same level as (1), i.e., character by character, but obeying a (simple) collating sequence that ensures that, for example, upper- and lowercase letters are equivalent, or that national characters are sorted correctly. The characters still do not have any “implicit” meaning, and the comparison of two strings is still deterministic; i.e., strings that are different at level 1 are still different at level 2.
- (3) *Text/application level.* Here we want to be able to search in text looking for specific words or items. The comparison is still performed on a character-by-character basis, but possibly ignoring some characters that are not important, and determinism is not important either.
- (4) *Semantic/dictionary/library/phone-book level.* Entire words like “the” are omitted from comparisons; maybe soundex is required. This probably requires specially developed software.

Our problem has been the conflicting requirements from each of these levels, which we optimistically have tried to combine into a single national locale (ignoring level 4, however). The POSIX Locale is aimed at level 2; i.e., at a rather low level. Many of our attempts to write a national Danish locale have failed because we have actually tried to write a level 3 locale, and finding that it did not work as

an alternative to the default POSIX locale at level 2.

The locale we now provide is the final compromise between level 2 and level 3, by taking our latest attempt aimed at level 3, and make the comparison completely deterministic, and thus bring it down to level 2.

We also have found that we may need to include some more information in the identification of a specific locale than just the country code, the language code, and the coded character set, since what we have had most problems with was the purpose or scope of a specific locale; i.e., is it just a nationalized version of the POSIX Locale (e.g., extended with `<ae>`, `<o/>`, and `<aa>` at the proper positions), is it aimed at text search (ignoring certain characters), or is it on an even higher level? Many such alternative locales would certainly be useful for various classes of problems or applications, so our model for the locale name identification string includes a `<version>` parameter.

We hope by providing these comments to have clarified our intention with the locale definitions to save other countries from doing our mistakes all over.

F.2 Locale String Definition Guideline

The following guideline is used for specifying the locale identification string:²⁾

```
"%2.2s_%2.2s.%s,%s",<language>,<territory>,<coded-character-
set>,<version>
```

where `<language>` shall be taken from ISO 639 {B1} and `<territory>` shall be the two-letter country code of ISO 3166 {B4}, if possible. The `<language>` shall be specified with lowercase letters only, and the `<territory>` shall be specified in uppercase letters only. An optional `<coded-character-set>` specification may follow after a `<period>` for the name of the coded character set; if just a numeric specification is present, this shall represent the number of the international standard describing the coded character set. If the `<coded-character-set>` specification is not present, the encoded character-set-specific locale shall be determined by the **CHARSET** environment variable, and if this is unset or null, the encoding of ISO 8859-1 {5} shall be assumed. A parameter specifying a `<version>` of the locale may be placed after the optional `<coded-character-set>` specification, delimited by `<comma>`. This may be used to discriminate between different cultural needs; for instance, dictionary order versus a more systems-oriented collating order.

²⁾ The guideline was inspired by the *X/Open Portability Guide* {B31}.

99 F.3 Scope of Danish National Locale

100 This national locale covers the Danish language in Denmark. In addition,
101 Faroese and Greenlandic `LC_TIME` and `LC_MESSAGES` specifications have been
102 defined; the rest of the Danish national locale shall be used for these locales as
103 well.

104 This locale is designed to be coded character-set independent. It completely
105 specifies the behavior of systems based on ISO/IEC 10646 {B11} (with ISO 6429
106 {B5} control character encoding) together with many 7-bit and 8-bit encoded char-
107 acter sets, including ISO 8859 character sets and major vendor-specific 8-bit char-
108 acter sets (with ISO 6429 {B5} or ISO/IEC 646 {1} control character encoding when
109 applicable).

110 This locale is portable as long as the character naming in the charmap description
111 file `ISO_10646` for ISO/IEC 10646 {B11} is followed. Examples of such charmap
112 files for ISO/IEC 10646 {B11} and ISO 8859-1 {5} are shown in F.5.1 and F.5.2.

113 The collating sequence is completely deterministic and is aimed for usage in sys-
114 tem tools. Other Danish collation sequences with nondeterministic properties,
115 which may be needed for some application programs, are not covered by this
116 locale.

117 The `LC_TYPE` category of the locale is quite general and may be useful for other
118 locales; also the `LC_COLLATE` category, though specifically Danish, may be a good
119 template from which to generate other locales.

120 Following the preceding guidelines for locale names, the national Danish locale
121 string shall be:

122 `da_DK`

123 F.3.1 `da_DK` — (Example) Danish National Locale

```

124 escape_char      /
125 comment_char     %
126 % Danish example national locale for the language Danish
127 % Source: Danish Standards Association
128 % Revision 1.7 1991-05-07

129 LC_CTYPE
1
130 digit    <0>;<1>;<2>;<3>;<4>;<5>;<6>;<7>;<8>;<9>
1
131 xdigit   <0>;<1>;<2>;<3>;<4>;<5>;<6>;<7>;<8>;<9>;/
132          <A>;<B>;<C>;<D>;<E>;<F>;<a>;<b>;<c>;<d>;<e>;<f>
1
133 blank    <SP>;<HT>;<NS>
1
134 space    <SP>;<LF>;<VT>;<FF>;<CR>;<HT>;<NS>
1
135 upper    <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;/
136          <K>;<L>;<M>;<N>;<O>;<P>;<Q>;<R>;<S>;<T>;/
137          <U>;<V>;<W>;<X>;<Y>;<Z>;<A!>;<A'>;<A/>;<A?>;/
1
```

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

138		<A>;<AA>;<AE>;<C,>;<E!>;<E'>;<E/>;<E>;<I!>;<I'>;/	1
139		<I/>;<I:>;<D->;<N?>;<O!>;<O'>;<O/>;<O?>;<O>;<O//>;/	1
140		<U!>;<U'>;<U/>;<U:>;<Y'>;<TH>;<A->;<C/>;<C,>;<E->;/	1
141		<E,>;<G/>;<G(>;<G,>;<G,>;<H/>;<I?>;<I->;<I,>;<I>;/	1
142		<J/>;<K,>;<H//>;<IJ>;<L,>;<L,>;<N,>;<OE>;<O->;<T//>;/	1
143		<NG>;<A>;<L//>;<L<>;<S'>;<S/>;<S<>;<S,>;<T<>;<Z'>;/	1
144		<Z<>;<Z,>;<R'>;<R,>;<A(>;<L'>;<C'>;<C<>;<E>;<E<>;/	1
145		<D<>;<D//>;<N'>;<N<>;<U?>;<O">;<U->;<U(>;<R<>;<U0>;/	1
146		<U>;<U">;<W/>;<Y/>;<T,>;<Y:>;<A<>;<A_>;<'A>;<A1>;/	
147		<A2>;<A3>;<B,>;<B_>;<D_>;<D,>;<D>;<E(>;<E_>;<E?>;/	
148		<F,>;<G<>;<G->;<G//>;<H:>;<H,>;<H,>;<H>;<I<>;<I(>;/	
149		<J(>;<K'>;<K<>;<K_>;<K,>;<K>;<L_>;<M'>;<M,>;<N,>;/	
150		<N_>;<O<>;<O(>;<O_>;<O>;<O1>;<P'>;<R,>;<R_>;<S,>;/	
151		<S>;<T_>;<T,>;<U<>;<V?>;<W'>;<W,>;<W>;<X,>;<X>;/	
152		<Y!>;<Y,>;<Z/>;<Z(>;<Z_>;<Z//>;<EZ>;<G'>;<'B>;<'D>;/	
153		<'G>;<'J>;<'Y>;<ED>;<IO>;<D%>;<G%>;<IE>;<DS>;<II>;/	
154		<YI>;<J%>;<LJ>;<NJ>;<Ts>;<KJ>;<V%>;<DZ>;<A=>;<B=>;/	
155		<V=>;<G=>;<D=>;<E=>;<Z%>;<Z=>;<I=>;<J=>;<K=>;<L=>;/	
156		<M=>;<N=>;<O=>;<P=>;<R=>;<S=>;<T=>;<U=>;<F=>;<H=>;/	
157		<C=>;<C%>;<S%>;<Sc>;<=>;<Y=>;<%>;<JE>;<JU>;<JA>;/	
158		<I3>;<A%>;<E%>;<Y%>;<I%>;<O%>;<U%>;<W%>;<A*>;<B*>;/	
159		<G*>;<D*>;<E*>;<Z*>;<Y*>;<H*>;<I*>;<K*>;<L*>;<M*>;/	
160		<N*>;<C*>;<O*>;<P*>;<R*>;<S*>;<T*>;<U*>;<F*>;<X*>;/	
161		<Q*>;<W*>;<J*>;<V*>	
162	lower	<a>;;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;/	
163		<k>;<l>;<m>;<n>;<o>;<p>;<q>;<r>;<s>;<t>;/	
164		<u>;<v>;<w>;<x>;<y>;<z>;<ss>;<a!>;<a'>;<a/>;/	
165		<a?>;<a:>;<aa>;<ae>;<c,>;<e!>;<e'>;<e/>;<e>;<i!>;/	
166		<i'>;<i/>;<i:>;<d->;<n?>;<o!>;<o'>;<o/>;<o?>;<o>;/	
167		<o//>;<u!>;<u'>;<u/>;<u:>;<y'>;<th>;<y:>;<a->;<c/>;/	
168		<c,>;<e->;<e,>;<g/>;<g(>;<g,>;<g,>;<h/>;<i?>;<i->;/	
169		<'n>;<kk>;<i>;<j/>;<k,>;<h//>;<i,>;<ij>;<l,>;<l,>;/	
170		<n,>;<oe>;<o->;<t//>;<ng>;<a>;<l//>;<l<>;<s'>;<s/>;/	
171		<s<>;<s,>;<t<>;<z'>;<z<>;<z,>;<r'>;<r,>;<a(>;<l'>;/	
172		<c'>;<c<>;<e>;<e<>;<d<>;<d//>;<n'>;<n<>;<u?>;<o">;/	
173		<u->;<u(>;<r<>;<u0>;<u>;<u">;<w/>;<y/>;<t,>;<a<>;/	1
174		<a_>;<'a>;<a1>;<a2>;<a3>;<b,>;<b_>;<d_>;<d,>;<d>;/	
175		<e>;<e_>;<e?>;<f,>;<g<>;<g->;<g//>;<h:>;<h,>;<h,>;/	
176		<h>;<i<>;<i(>;<j(>;<k'>;<k<>;<k_>;<k,>;<k>;<l_>;/	
177		<m'>;<m,>;<n,>;<n_>;<o<>;<o(>;<o_>;<o>;<o1>;<p'>;/	
178		<r,>;<r_>;<s,>;<s>;<t_>;<t,>;<u<>;<v?>;<w'>;<w,>;/	
179		<w:>;<x,>;<x>;<y!>;<y,>;<z/>;<z(>;<z_>;<z//>;<ez>;/	
180		<g'>;<'b>;<'d>;<'g>;<'j>;<'y>;<ed>;<nS>;<sB>;<a=>;/	
181		<b=>;<v=>;<g=>;<d=>;<e=>;<z%>;<z=>;<i=>;<j=>;<k=>;/	
182		<l=>;<m=>;<n=>;<o=>;<p=>;<r=>;<s=>;<t=>;<u=>;<f=>;/	
183		<h=>;<c=>;<c%>;<s%>;<sc>;<=>;<y=>;<%>;<j>;<ju>;/	
184		<ja>;<io>;<d%>;<g%>;<ie>;<ds>;<ii>;<yi>;<j%>;<l>;/	
185		<nj>;<ts>;<kj>;<v%>;<dz>;<a%>;<e%>;<y%>;<i%>;<a*>;/	
186		<b*>;<g*>;<d*>;<e*>;<z*>;<y*>;<h*>;<i*>;<k*>;<l*>;/	
187		<m*>;<n*>;<c*>;<o*>;<p*>;<r*>;<s*>;<s*>;<t*>;<u*>;/	
188		<f*>;<x*>;<q*>;<w*>;<j*>;<v*>;<o%>;<u%>;<w%>;<A5>;/	
189		<I5>;<U5>;<E5>;<O5>;<tU>;<yA>;<yU>;<yO>;<wA>;<a6>;/	
190		<i6>;<u6>;<e6>;<o6>;<TU>;<YA>;<YU>;<YO>;<WA>;<KA>;/	
191		<KE>;<ff>;<fi>;<fl>;<ft>;<st>	
192	alpha	<A>;;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;/	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

```

193 <K>;<L>;<M>;<N>;<O>;<P>;<Q>;<R>;<S>;<T>; /
194 <U>;<V>;<W>;<X>;<Y>;<Z>;<a>;<b>;<c>;<d>; /
195 <e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;<n>; /
196 <o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>; /
197 <y>;<z>;<-->;<A!>;<A'>;<A/>;<A?>;<A:>;<AA>;<AE>; /
198 <C,>;<E!>;<E'>;<E/>;<E:>;<E!>;<E'>;<E/>;<E:>;<D->; /
199 <N?>;<O!>;<O'>;<O/>;<O:>;<O!>;<O'>;<O/>;<O:>;<U!>;<U'>;<U/>; /
200 <U:>;<Y'>;<TH>;<ss>;<a!>;<a'>;<a/>;<a?>;<a:>;<aa>; /
201 <ae>;<c,>;<e!>;<e'>;<e/>;<e:>;<e!>;<e'>;<e/>;<e:>; /
202 <d->;<n?>;<o!>;<o'>;<o/>;<o:>;<o!>;<o'>;<o/>;<o:>; /
203 <u/>;<u:>;<y'>;<th>;<y:>;<A->;<C/>;<C,>;<E->;<E,>; /
204 <G/>;<G:>;<a->;<c/>;<c,>;<e->;<e,>;<g/>;<g:>;<G,>; /
205 <G,>;<H/>;<I?>;<I->;<I,>;<g,>;<g,>;<h/>;<i?>;<i->; /
206 <I>;<J/>;<K,>;<H/>;<IJ>;<L,>;<L,>;<N,>;<OE>;<O->; /
207 <T//>;<NG>;<'n>;<kk>;<i>;<j/>;<k,>;<h/>;<i>;<ij>; /
208 <l,>;<l,>;<n,>;<oe>;<o->;<t//>;<ng>;<A>;<L/>;<L>; /
209 <S'>;<S/>;<S>;<S>;<T>;<Z'>;<Z>;<Z,>;<a>;<l/>; /
210 <l>;<s'>;<s/>;<s>;<s>;<t>;<z'>;<z>;<z,>;<R'>; /
211 <R,>;<A>;<L'>;<C'>;<C>;<E>;<E>;<D>;<D/>;<N'>; /
212 <N>;<U?>;<O">;<U->;<U>;<R>;<U0>;<U>;<U">;<W/>; /
213 <Y/>;<T,>;<Y:>;<r'>;<r,>;<a>;<l'>;<c'>;<c>;<e>; /
214 <e>;<d>;<d/>;<n'>;<n>;<u?>;<o">;<u->;<u>;<r>; /
215 <u0>;<u>;<u">;<w/>;<y/>;<t,>;<a>;<A>;<a>;<A>; /
216 <'a>;<'A>;<a1>;<A1>;<a2>;<A2>;<a3>;<A3>;<b,>;<B,>; /
217 <b_>;<B_>;<d_>;<D_>;<d,>;<D,>;<d>;<D>;<e>;<E>; /
218 <e_>;<E_>;<e?>;<E?>;<f,>;<F,>;<g>;<G>;<g->;<G->; /
219 <g//>;<G//>;<h:>;<H:>;<h,>;<H,>;<h,>;<H,>;<h>;<H>; /
220 <i>;<I>;<i>;<I>;<j>;<J>;<k'>;<K'>;<k>;<K>; /
221 <k_>;<K_>;<k,>;<K,>;<k>;<K>;<l_>;<L_>;<m'>;<M'>; /
222 <m,>;<M,>;<n,>;<N,>;<n_>;<N_>;<o>;<O>;<o>;<O>; /
223 <o_>;<O_>;<o>;<O>;<o1>;<O1>;<p'>;<P'>;<r,>;<R,>; /
224 <r_>;<R_>;<s,>;<S,>;<s>;<S>;<t_>;<T_>;<t,>;<T,>; /
225 <u>;<U>;<v?>;<V?>;<w'>;<W'>;<w,>;<W,>;<w>;<W>; /
226 <x,>;<X,>;<x>;<X>;<y!>;<Y!>;<y,>;<Y,>;<z/>;<Z/>; /
227 <z>;<Z>;<z>;<Z>;<z/>;<Z/>;<ez>;<EZ>;<g'>;<G'>; /
228 <'b>;<'B>;<'d>;<'D>;<'g>;<'G>;<'j>;<'J>;<'y>;<'Y>; /
229 <ed>;<ED>;<nS>;<IO>;<D%>;<G%>;<IE>;<DS>;<II>;<YI>; /
230 <J%>;<LJ>;<NJ>;<Ts>;<KJ>;<V%>;<DZ>;<A=>;<B=>;<V=>; /
231 <G=>;<D=>;<E=>;<Z%>;<Z=>;<I=>;<J=>;<K=>;<L=>;<M=>; /
232 <N=>;<O=>;<P=>;<R=>;<S=>;<T=>;<U=>;<F=>;<H=>;<C=>; /
233 <C%>;<S%>;<Sc>;<=>;<Y=>;<%>;<JE>;<JU>;<JA>;<a=>; /
234 <b=>;<v=>;<g=>;<d=>;<e=>;<z%>;<z=>;<i=>;<j=>;<k=>; /
235 <l=>;<m=>;<n=>;<o=>;<p=>;<r=>;<s=>;<t=>;<u=>;<f=>; /
236 <h=>;<c=>;<c%>;<s%>;<sc>;<='>;<y=>;<%>;<je>;<ju>; /
237 <ja>;<io>;<d%>;<g%>;<ie>;<ds>;<ii>;<yi>;<j%>;<lj>; /
238 <nj>;<ts>;<kj>;<v%>;<dz>;<I3>;<A%>;<E%>;<Y%>;<I%>; /
239 <O%>;<U%>;<W%>;<A*>;<B*>;<G*>;<D*>;<E*>;<Z*>;<Y*>; /
240 <H*>;<I*>;<K*>;<L*>;<M*>;<N*>;<C*>;<O*>;<P*>;<R*>; /
241 <S*>;<T*>;<U*>;<F*>;<X*>;<Q*>;<W*>;<J*>;<V*>;<a%>; /
242 <e%>;<y%>;<i%>;<a*>;<b*>;<g*>;<d*>;<e*>;<z*>;<y*>; /
243 <h*>;<i*>;<k*>;<l*>;<m*>;<n*>;<c*>;<o*>;<p*>;<r*>; /
244 <*s>;<s*>;<t*>;<u*>;<f*>;<x*>;<q*>;<w*>;<j*>;<v*>; /
245 <o%>;<u%>;<w%>;<p>;<v>;<gf>;<H'>;<aM>;<aH>;<wH>; /
246 <ah>;<yH>;<a+>;<b+>;<tm>;<t+>;<tk>;<g+>;<hk>;<x+>; /
247 <d+>;<dk>;<r+>;<z+>;<s+>;<sn>;<c+>;<dd>;<tj>;<zH>; /
248 <e+>;<i+>;<f+>;<q+>;<k+>;<l+>;<m+>;<n+>;<h+>;<w+>; /
249 <j+>;<y+>;<A+>;<B+>;<G+>;<D+>;<H+>;<W+>;<Z+>;<X+>; /

```

1

1

```

250      <Tj>;<J+>;<K%>;<K+>;<L+>;<M%>;<M+>;<N%>;<N+>;<S+>;/
251      <E+>;<P%>;<P+>;<Zj>;<ZJ>;<Q+>;<R+>;<Sh>;<T+>;<b4>;/
252      <p4>;<m4>;<f4>;<d4>;<t4>;<n4>;<l4>;<g4>;<k4>;<h4>;/
253      <j4>;<q4>;<x4>;<zh>;<ch>;<sh>;<r4>;<z4>;<c4>;<s4>;/
254      <a4>;<o4>;<e4>;<eh>;<ai>;<ei>;<au>;<ou>;<an>;<en>;/
255      <aN>;<eN>;<er>;<i4>;<u4>;<iu>;<A5>;<a5>;<I5>;<i5>;/
256      <U5>;<u5>;<E5>;<e5>;<O5>;<o5>;<ka>;<ga>;<ki>;<gi>;/
257      <ku>;<gu>;<ke>;<ge>;<ko>;<go>;<sa>;<za>;<si>;<zi>;/
258      <su>;<zu>;<se>;<ze>;<so>;<zo>;<ta>;<da>;<ti>;<di>;/
259      <tU>;<tu>;<du>;<te>;<de>;<to>;<do>;<na>;<ni>;<nu>;/
260      <ne>;<no>;<ha>;<ba>;<pa>;<hi>;<bi>;<pi>;<hu>;<bu>;/
261      <pu>;<he>;<be>;<pe>;<ho>;<bo>;<po>;<ma>;<mi>;<mu>;/
262      <me>;<mo>;<yA>;<ya>;<yU>;<yu>;<yO>;<yo>;<ra>;<ri>;/
263      <ru>;<re>;<ro>;<wA>;<wa>;<wi>;<we>;<wo>;<n5>;<a6>;/
264      <A6>;<i6>;<I6>;<u6>;<U6>;<e6>;<E6>;<o6>;<O6>;<Ka>;/
265      <Ga>;<Ki>;<Gi>;<Ku>;<Gu>;<Ke>;<Ge>;<Ko>;<Go>;<Sa>;/
266      <Za>;<Si>;<Zi>;<Su>;<Zu>;<Se>;<Ze>;<So>;<Zo>;<Ta>;/
267      <Da>;<Ti>;<Di>;<TU>;<Tu>;<Du>;<Te>;<De>;<To>;<Do>;/
268      <Na>;<Ni>;<Nu>;<Ne>;<No>;<Ha>;<Ba>;<Pa>;<Hi>;<Bi>;/
269      <Pi>;<Hu>;<Bu>;<Pu>;<He>;<Be>;<Pe>;<Ho>;<Bo>;<Po>;/
270      <Ma>;<Mi>;<Mu>;<Me>;<Mo>;<YA>;<Ya>;<YU>;<Yu>;<YO>;/
271      <Yo>;<Ra>;<Ri>;<Ru>;<Re>;<Ro>;<WA>;<Wa>;<Wi>;<We>;/
272      <Wo>;<N6>;<Vu>;<KA>;<KE>;<ff>;<fi>;<fl>;<ft>;<st>;/
273      <yf>

274      cntrl  <NU>;<SH>;<SX>;<EX>;<ET>;<EQ>;<AK>;<BL>;<BS>;<HT>;/
275      <LF>;<VT>;<FF>;<CR>;<SO>;<SI>;<DL>;<D1>;<D2>;<D3>;/
276      <D4>;<NK>;<SY>;<EB>;<CN>;<EM>;<SB>;<EC>;<FS>;<GS>;/
277      <RS>;<US>;<DT>;<PA>;<HO>;<BH>;<NH>;<IN>;<NL>;<SA>;/
278      <ES>;<HS>;<HJ>;<VS>;<PD>;<PU>;<RI>;<S2>;<S3>;<DC>;/
279      <P1>;<P2>;<TS>;<CC>;<MW>;<SG>;<EG>;<SS>;<GC>;<SC>;/
280      <CI>;<ST>;<OC>;<PM>;<AC>

281      punct  <!>;<">;<Nb>;<DO>;<%>;<&>;<'>;<( )>;<*>;/
282      <+>;<,>;<->;<.>;< />;<:>;<:>;<=>;< />;>;/
283      <?>;<At>;<( )>;< />;>;<'>;<_>;<'!>;<(!>;<!!>;/
284      <!)>;<'?'>;<!I>;<Ct>;<Pd>;<Cu>;<Ye>;<BB>;<SE>;<'>;/
285      <Co>;<-a>;<=>;<NO>;<Rg>;<'>;<DG>;<+->;<2S>;<3S>;/
286      <'>;<My>;<PI>;<.M>;<'>;<1S>;<-o>;< />;>;<14>;<12>;/
287      <34>;<?I>;<*X>;<->;<'6>;<6>;<=>;<-!>;<->;<-v>;/
288      <'9>;<"9>;<'0>;<HB>;<TM>;<Md>;<18>;<38>;<58>;<78>;/
289      <Om>;<'>;<'>;<'>;<'>;<'>;<'>;<S>;<Vs>;<1M>;<1N>;/
290      <3M>;<4M>;<6M>;<1H>;<1T>;<-1>;<-N>;<-2>;<-M>;<-3>;/
291      <'1>;<'2>;<'3>;<9'>;<9">;<.9>;<:9>;<<1>;< />;>;< />;/
292      < />;>;<15>;<25>;<35>;<45>;<16>;<13>;<23>;<56>;<*->;/
293      < />;>;< />;>;<-X>;<%0>;<co>;<PO>;<Rx>;<AO>;<oC>;<M1>;/
294      <Fm>;<T1>;<TR>;<MX>;<Mb>;<Mx>;<XX>;<OK>;<M2>;<12>;/
295      <=2>;<Ca>;<. .>;<.3>;<:3>;<. :>;<. :>;<-+>;<!=>;<=3>;/
296      <?1>;<?2>;<?->;<?=>;<=>;< />;>;<0>;<00>;<PP>;<-T>;/
297      <-L>;<-V>;<AN>;<OR>;<.P>;<dP>;<f>;<In>;<Io>;<RT>;/
298      <*P>;<+Z>;<FA>;<TE>;<GF>;<DE>;<NB>;<(U)>;<(C)>;/
299      <)C>;<(_>;<_>;<(->;<->;< />;>;<UD>;<Ub>;<=>;<=>;/
300      <==>;< />;>;<OL>;<0u>;<0U>;<SU>;<0>;<OS>;<fS>;<Or>;/
301      <SR>;<uT>;<UT>;<dT>;<Dt>;<PL>;<PR>;<*1>;<*2>;<VV>;/
302      <HH>;<DR>;<LD>;<UR>;<UL>;<VR>;<VL>;<DH>;<UH>;<VH>;/
303      <TB>;<LB>;<FB>;<sB>;<EH>;<vv>;<hh>;<dr>;<dl>;<ur>;/
304      <ul>;<vr>;<vl>;<dh>;<uh>;<vh>;<.S>;<:S>;<?S>;<1B>;/

```

1
1

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

```

305 <RB>;<C>;<CD>;<Dr>;<DL>;<Ur>;<UL>;<Vr>;<VL>;<dH>;//
306 <uH>;<vH>;<Ob>;<Sb>;<Sn>;<Pt>;<NI>;<CH>;<CS>;<dR>;//
307 <dL>;<uR>;<uL>;<vR>;<vL>;<Dh>;<Uh>;<Vh>;<0m>;<0M>;//
308 <Ic>;<SM>;<CG>;<Ci>;<(A>;</>V>;<!>;<*>;<!/>;<*/>;//
309 <<7>;<7>;</>7>;<7/>;<I2>;<0.>;<HI>;<::>;<FD>;<LZ>;//
310 <BD>;<1R>;<2R>;<3R>;<4R>;<5R>;<6R>;<7R>;<8R>;<9R>;//
311 <aR>;<bR>;<cR>;<N0>;<i3>;<ii>;<,,>;<!*>;<?*>;<!'>;//
312 <,>;<!'>;<!,>;<?>;<?,>;<!:>;<?>;<'>;<,>;<+>;<+>;//
313 <?+>;<+>;<+>;<+>;<+>;<+>;<+>;<+>;<+>;<+>;<+>;//
314 <IS>;<,>;<,>;<,>;<,>;<,>;<,>;<,>;<,>;<,>;<,>;//
315 <<'>;</>'>;<,>;<,>;<,>;<,>;<,>;<,>;<,>;<,>;<,>;//
316 <KM>;<"5>;<05>;<*5>;<+5>;<-6>;<*6>;<+6>;<Iu>;<Il>;//
317 <__>;<"!>;<">;<">;<">;<">;<">;<">;<">;<">;//
318 <"0>;<">;<">;<">;<">;<">;<">;<">;<">;<">;//
319 <Bd>;<Fl>;<Li>;</f>;<0s>;<1s>;<2s>;<3s>;<4s>;<5s>;//
320 <6s>;<7s>;<8s>;<9s>;<0S>;<4S>;<5S>;<6S>;<7S>;<8S>;//
321 <9S>;<+S>;<-S>;<1h>;<2h>;<3h>;<4h>;<1j>;<2j>;<3j>;//
322 <4j>;<UA>;<UB>;<yr>;<.6>;<6>;</>6>;<,>;<6>;<6>;<S>;//
323 <)>S>

324 tolower (<'A>,<'a>); (<'B>,<'b>); (<'D>,<'d>); (<'G>,<'g>); (<'J>,<'j>);//
325 (<'Y>,<'y>); (<A>,<a>); (<A!>,<a!>); (<A'>,<a'>); (<A(>,<a(>);//
326 (<A->,<a->); (<A1>,<a1>); (<A2>,<a2>); (<A3>,<a3>); (<A:>,<a:>);//
327 (<A;>,<a;>); (<A<>,<a<>); (<A/>,<a/>); (<A?>,<a?>); (<AA>,<aa>);//
328 (<AE>,<ae>); (<A_>,<a_>); (<B>,<b>); (<B.>,<b.>); (<B_>,<b_>);//
329 (<C>,<c>); (<C'>,<c'>); (<C,>,<c,>); (<C.>,<c.>); (<C<>,<c<>);//
330 (<C/>,<c/>); (<D>,<d>); (<D->,<d->); (<D.>,<d.>); (<D/>,<d/>);//
331 (<D;>,<d;>); (<D<>,<d<>); (<D_>,<d_>); (<E>,<e>); (<E!>,<e!>);//
332 (<E'>,<e'>); (<E(>,<e(>); (<E->,<e->); (<E.>,<e.>); (<E:>,<e:>);//
333 (<E;>,<e;>); (<E<>,<e<>); (<E/>,<e/>); (<E?>,<e?>); (<ED>,<ed>);//
334 (<EZ>,<ez>); (<E_>,<e_>); (<F>,<f>); (<F.>,<f.>);//
335 (<G>,<ft>); (<G'>,<g'>); (<G(>,<g(>); (<G,>,<g,>);//
336 (<G->,<g->); (<G.>,<g.>); (<G/>,<g/>); (<G<>,<g<>); (<G/>,<g/>);//
337 (<H>,<h>); (<H,>,<h,>); (<H.>,<h.>); (<H/>,<h/>); (<H:>,<h:>);//
338 (<H;>,<h;>); (<H/>,<h/>); (<I>,<i>); (<I!>,<i!>); (<I'>,<i'>);//
339 (<I(>,<i(>); (<I->,<i->); (<I.>,<i.>); (<I:>,<i:>); (<I;>,<i;>);//
340 (<I<>,<i<>); (<I/>,<i/>); (<I?>,<i?>); (<IJ>,<ij>); (<J>,<j>);//
341 (<J(>,<j(>); (<J/>,<j/>); (<K>,<k>); (<K'>,<k'>); (<K,>,<k,>);//
342 (<K.>,<k.>); (<K;>,<k;>); (<K<>,<k<>); (<K_>,<k_>);//
343 (<L>,<l>); (<L'>,<l'>); (<L,>,<l,>); (<L.>,<l.>); (<L/>,<l/>);//
344 (<L<>,<l<>); (<L_>,<l_>); (<M>,<m>); (<M'>,<m'>); (<M.>,<m.>);//
345 (<N>,<n>); (<N'>,<n'>); (<N,>,<n,>); (<N.>,<n.>); (<N<>,<n<>);//
346 (<N?>,<n?>); (<NG>,<ng>); (<N_>,<n_>); (<O>,<o>); (<O!>,<o!>);//
347 (<O">,<o">); (<O'>,<o'>); (<O(>,<o(>); (<O->,<o->); (<O/>,<o/>);//
348 (<O1>,<o1>); (<O:>,<o:>); (<O;>,<o;>); (<O<>,<o<>); (<O/>,<o/>);//
349 (<O?>,<o?>); (<OE>,<oe>); (<O_>,<o_>); (<P>,<p>); (<P'>,<p'>);//
350 (<Q>,<q>); (<R>,<r>); (<R'>,<r'>); (<R,>,<r,>); (<R.>,<r.>);//
351 (<R<>,<r<>); (<R_>,<r_>); (<S>,<s>); (<S'>,<s'>); (<S,>,<s,>);//
352 (<S.>,<s.>); (<S;>,<s;>); (<S<>,<s<>); (<S/>,<s/>); (<T>,<t>);//
353 (<T,>,<t>); (<T.>,<t.>); (<T/>,<t/>); (<T<>,<t<>);//
354 (<TH>,<th>); (<T_>,<t_>); (<U>,<u>); (<U!>,<u!>); (<U">,<u">);//
355 (<U'>,<u'>); (<U(>,<u(>); (<U->,<u->); (<U0>,<u0>); (<U:>,<u:>);//
356 (<U;>,<u;>); (<U<>,<u<>); (<U/>,<u/>); (<U?>,<u?>); (<V>,<v>);//
357 (<V?>,<v?>); (<W>,<w>); (<W'>,<w'>); (<W.>,<w.>); (<W:>,<w:>);//
358 (<W/>,<w/>); (<X>,<x>); (<X.>,<x.>); (<X:>,<x:>); (<Y>,<y>);//
359 (<Y!>,<y!>); (<Y'>,<y'>); (<Y.>,<y.>); (<Y:>,<y:>); (<Y/>,<y/>);//
360 (<Z>,<z>); (<Z'>,<z'>); (<Z(>,&lt
```

```

361      (<Z<>,<z<>); (<Z/>,<z/>); (<Z_>,<z_>); (<%>,<%>); (<=>,<=>); /
362      (<A=>,<a=>); (<B=>,<b=>); (<C%>,<c%>); (<C=>,<c=>); (<D%>,<d%>); /
363      (<D=>,<d=>); (<DS>,<ds>); (<DZ>,<dz>); (<E=>,<e=>); (<F=>,<f=>); /
364      (<G%>,<g%>); (<G=>,<g=>); (<H=>,<h=>); (<I=>,<i=>); (<IE>,<ie>); /
365      (<II>,<ii>); (<IO>,<io>); (<J%>,<j%>); (<J=>,<j=>); (<JA>,<ja>); /
366      (<JE>,<je>); (<JU>,<ju>); (<K=>,<k=>); (<KJ>,<kj>); (<L=>,<l=>); /
367      (<LJ>,<lj>); (<M=>,<m=>); (<N=>,<n=>); (<NJ>,<nj>); (<O=>,<o=>); /
368      (<P=>,<p=>); (<R=>,<r=>); (<S%>,<s%>); (<S=>,<s=>); (<Sc>,<sc>); /
369      (<T=>,<t=>); (<Ts>,<ts>); (<U=>,<u=>); (<V=>,<v=>); (<Y=>,<y=>); /
370      (<YI>,<yi>); (<Z%>,<z%>); (<Z=>,<z=>); (<A%>,<a%>); (<A*,>,<a*>); /
371      (<B*,>,<b*>); (<C*,>,<c*>); (<D*,>,<d*>); (<E%>,<e%>); (<E*,>,<e*>); /
372      (<F*,>,<f*>); (<G*,>,<g*>); (<H*,>,<h*>); (<I%>,<i%>); (<I*,>,<i*>); /
373      (<J*,>,<j*>); (<K*,>,<k*>); (<L*,>,<l*>); (<M*,>,<m*>); (<N*,>,<n*>); /
374      (<O%>,<o%>); (<O*,>,<o*>); (<P*,>,<p*>); (<Q*,>,<q*>); (<R*,>,<r*>); /
375      (<S*,>,<s*>); (<T*,>,<t*>); (<U%>,<u%>); (<U*,>,<u*>); (<V*,>,<v*>); /
376      (<W%>,<w%>); (<W*,>,<w*>); (<X*,>,<x*>); (<Y%>,<y%>); (<Y*,>,<y*>); /
377      (<Z*,>,<z*>)

378  toupper (<'a>,<'A>); (<'b>,<'B>); (<'d>,<'D>); (<'g>,<'G>); (<'j>,<'J>); /
379      (<'y>,<'Y>); (<a>,<A>); (<a!>,<A!>); (<a'>,<A'>); (<a(>,<A(>); /
380      (<a->,<A->); (<a1>,<A1>); (<a2>,<A2>); (<a3>,<A3>); (<a:>,<A:>); /
381      (<a;>,<A;>); (<a<>,<A<>); (<a/>,<A/>); (<a?>,<A?>); (<aa>,<AA>); /
382      (<ae>,<AE>); (<a_>,<A_>); (<b>,<B>); (<b.>,<B.>); (<b_>,<B_>); /
383      (<c>,<C>); (<c'>,<C'>); (<c,>,<C,>); (<c.>,<C.>); (<c<>,<C<>); /
384      (<c/>,<C/>); (<d>,<D>); (<d->,<D->); (<d.>,<D.>); (<d//>,<D//>); /
385      (<d!>,<D!>); (<d_>,<D_>); (<e>,<E>); (<e!>,<E!>); /
386      (<e'>,<E'>); (<e(>,<E(>); (<e->,<E->); (<e.>,<E.>); (<e<>,<E<>); /
387      (<e;>,<E;>); (<e<>,<E<>); (<e/>,<E/>); (<e?>,<E?>); (<ed>,<ED>); /
388      (<ez>,<EZ>); (<e_>,<E_>); (<f>,<F>); (<f.>,<F.>); /
389      (<ft>,<G>); (<g'>,<G'>); (<g(>,<G(>); (<g,>,<G,>); /
390      (<g->,<G->); (<g.>,<G.>); (<g//>,<G//>); (<g<>,<G<>); (<g/>,<G/>); /
391      (<h>,<H>); (<h,>,<H,>); (<h.>,<H.>); (<h//>,<H//>); (<h:>,<H:>); /
392      (<h;>,<H;>); (<h/>,<H/>); (<i>,<I>); (<i!>,<I!>); (<i'>,<I'>); /
393      (<i(>,<I(>); (<i->,<I->); (<i.>,<I.>); (<i:>,<I:>); (<i;>,<I;>); /
394      (<i<>,<I<>); (<i/>,<I/>); (<i?>,<I?>); (<ij>,<IJ>); (<j>,<J>); /
395      (<j(>,<J(>); (<j/>,<J/>); (<k>,<K>); (<k'>,<K'>); (<k,>,<K,>); /
396      (<k.>,<K.>); (<k!>,<K!>); (<k<>,<K<>); (<k_>,<K_>); /
397      (<l>,<L>); (<l'>,<L'>); (<l,>,<L,>); (<l.>,<L.>); (<l//>,<L//>); /
398      (<l_>,<L_>); (<l_>,<L_>); (<m>,<M>); (<m'>,<M'>); (<m.>,<M.>); /
399      (<n>,<N>); (<n'>,<N'>); (<n,>,<N,>); (<n.>,<N.>); (<n<>,<N<>); /
400      (<n?>,<N?>); (<ng>,<NG>); (<n_>,<N_>); (<o>,<O>); (<o!>,<O!>); /
401      (<o">,<O">); (<o'>,<O'>); (<o(>,<O(>); (<o->,<O->); (<o//>,<O//>); /
402      (<o1>,<O1>); (<o:>,<O:>); (<o;>,<O;>); (<o<>,<O<>); (<o/>,<O/>); /
403      (<o?>,<O?>); (<oe>,<OE>); (<o_>,<O_>); (<p>,<P>); (<p'>,<P'>); /
404      (<q>,<Q>); (<r>,<R>); (<r'>,<R'>); (<r,>,<R,>); (<r.>,<R.>); /
405      (<r<>,<R<>); (<r_>,<R_>); (<s>,<S>); (<s'>,<S'>); (<s,>,<S,>); /
406      (<s.>,<S.>); (<s!>,<S!>); (<s<>,<S<>); (<s/>,<S/>); (<st>,<T>); /
407      (<t>,<T>); (<t.>,<T.>); (<t//>,<T//>); (<t<>,<T<>); /
408      (<th>,<TH>); (<t_>,<T_>); (<u>,<U>); (<u!>,<U!>); (<u">,<U">); /
409      (<u'>,<U'>); (<u(>,<U(>); (<u->,<U->); (<u0>,<U0>); (<u:>,<U:>); /
410      (<u;>,<U;>); (<u<>,<U<>); (<u/>,<U/>); (<u?>,<U?>); (<v>,<V>); /
411      (<v?>,<V?>); (<w>,<W>); (<w'>,<W'>); (<w.>,<W.>); (<w:>,<W:>); /
412      (<w/>,<W/>); (<x>,<X>); (<x.>,<X.>); (<x:>,<X:>); (<y>,<Y>); /
413      (<y!>,<Y!>); (<y'>,<Y'>); (<y.>,<Y.>); (<y:>,<Y:>); (<y/>,<Y/>); /
414      (<z>,<Z>); (<z'>,<Z'>); (<z(>,<Z(>); (<z.>,<Z.>); (<z//>,<Z//>); /
415      (<z<>,<Z<>); (<z/>,<Z/>); (<z_>,<Z_>); (<%>,<%>); (<=>,<=>); /
416      (<a=>,<A=>); (<b=>,<B=>); (<c%>,<C%>); (<c=>,<C=>); (<d%>,<D%>); /

```

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.


```

417      (<d=>,<D=>); (<ds>,<DS>); (<dz>,<DZ>); (<e=>,<E=>); (<f=>,<F=>); /
418      (<g>,<G>); (<g=>,<G=>); (<h=>,<H=>); (<i=>,<I=>); (<ie>,<IE>); /
419      (<ii>,<II>); (<io>,<IO>); (<j>,<J>); (<j=>,<J=>); (<ja>,<JA>); /
420      (<je>,<JE>); (<ju>,<JU>); (<k>,<K>); (<kj>,<KJ>); (<l>,<L>); /
421      (<lj>,<LJ>); (<m>,<M>); (<n>,<N>); (<nj>,<NJ>); (<o>,<O>); /
422      (<p>,<P>); (<r>,<R>); (<s>,<S>); (<s=>,<S=>); (<sc>,<Sc>); /
423      (<t>,<T>); (<ts>,<Ts>); (<u>,<U>); (<v>,<V>); (<y>,<Y>); /
424      (<yi>,<YI>); (<z>,<Z>); (<z=>,<Z=>); (<a>,<A>); (<a*>,<A*>); /
425      (<b>,<B>); (<c>,<C>); (<d>,<D>); (<e>,<E>); (<e*>,<E*>); /
426      (<f>,<F>); (<g>,<G>); (<h>,<H>); (<i>,<I>); (<i*>,<I*>); /
427      (<j>,<J>); (<k>,<K>); (<l>,<L>); (<m>,<M>); (<n>,<N>); /
428      (<o>,<O>); (<o*>,<O*>); (<p>,<P>); (<q>,<Q>); (<r>,<R>); /
429      (<s>,<S>); (<s*>,<S*>); (<t>,<T>); (<u>,<U>); (<u*>,<U*>); /
430      (<v>,<V>); (<w>,<W>); (<w*>,<W*>); (<x>,<X>); (<y>,<Y>); /
431      (<y*>,<Y*>); (<z>,<Z>);
1

432  END LC_CTYPE

433  LC_COLLATE

434  % Ordering algorithm:
435  % 1. Spaces and hyphen (but not soft hyphen) before punctuation
436  % characters, punctuation characters before numbers,
437  % numbers before letters.
438  % 2. Letters with diacritical marks are members of equivalence classes
439  % 3. Upper case letters before corresponding lower case letter.
440  % 4. Specials are ignored when comparing letters, but then they are considered
441  % 5. The alphabets are sorted in the order of appearance in ISO 10646:
442  % Latin, Cyrillic, Greek, Arabic and Hebrew.
443  % 6. In Danish, the letter combination 'aa' is equivalent to '<aa>'
444  %
445  % The ordering algorithm is in accordance with Danish Standard DS 377
446  % and the Danish Orthography Dictionary (Retskrivningsordbogen, 1986).
447  % It is also in accordance with Faroese and Greenlandic orthography.
1

448  collating-element <A-A> from <A><A>
449  collating-element <a-a> from <a><a>
450  collating-element <A-a> from <A><a>
451  collating-element <s-s> from <s><s>
452  collating-element <i-j> from <i><j>
453  collating-element <I-J> from <I><J>
454  collating-element <o-e> from <o><e>
455  collating-element <O-E> from <O><E>
456  collating-element <t-h> from <t><h>
457  collating-element <T-H> from <T><H>
458  collating-element <n-g> from <n><g>
459  collating-element <N-G> from <N><G>
1

460  % collating symbols, <CAPITAL> or <SMALL> letters first
461  % <CAPITAL> letters before <SMALL> letters
1

462  collating-symbol <CAPITAL>
463  collating-symbol <BOTH>
464  collating-symbol <SMALL>

465  collating-symbol <NO-ACCENT>
466  collating-symbol <ACUTE>

```

```

467 collating-symbol <GRAVE>
468 collating-symbol <CIRCUMFLEX>
469 collating-symbol <TILDE>
470 collating-symbol <MACRON>
471 collating-symbol <BREVE>
472 collating-symbol <DOT>
473 collating-symbol <DIAERESIS>
474 collating-symbol <CEDILLA>
475 collating-symbol <UNDERLINE>
476 collating-symbol <STROKE>
477 collating-symbol <DOUBLE-ACUTE>
478 collating-symbol <OGONEK>
479 collating-symbol <CARON>
480 collating-symbol <CYRILLIC>
481 collating-symbol <GREEK>
482 collating-symbol <ALPHA-1>
483 collating-symbol <ALPHA-2>
484 collating-symbol <PRECEDED-BY-APOSTROPHE>
485 collating-symbol <SPECIAL>
486 collating-symbol <ACC0>
487 collating-symbol <ACC1>
488 collating-symbol <ACC2>
489 collating-symbol <ACC3>
490 collating-symbol <ACC11>
491 collating-symbol <ACC12>

492 % letter;accent;case;specials

493 order_start forward;backward;forward;forward

494 <CAPITAL>
495 <BOTH>
496 <SMALL>

497 <NO-ACCENT>
498 <ACUTE>
499 <GRAVE>
500 <CIRCUMFLEX>
501 <TILDE>
502 <MACRON>
503 <BREVE>
504 <DOT>
505 <DIAERESIS>
506 <CEDILLA>
507 <UNDERLINE>
508 <STROKE>
509 <DOUBLE-ACUTE>
510 <OGONEK>
511 <CARON>
512 <CYRILLIC>
513 <GREEK>
514 <ALPHA-1>
515 <ALPHA-2>
516 <PRECEDED-BY-APOSTROPHE>
517 <SPECIAL>
518 <ACC0>
519 <ACC1>

```

1

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

520	<ACC2>	
521	<ACC3>	
522	<ACC11>	
523	<ACC12>	
524	<SP>	<SP>
525	<NS>	<SP>
526	<HT>	<SP>
527	<VT>	<SP>
528	<CR>	<SP>
529	<LF>	<SP>
530	<FF>	<SP>
531	<->	<SP>
532	</ />	<SP>
533	<!>	IGNORE; IGNORE; IGNORE
534	<">	IGNORE; IGNORE; IGNORE
535	<Nb>	IGNORE; IGNORE; IGNORE
536	<DO>	IGNORE; IGNORE; IGNORE
537	<%>	IGNORE; IGNORE; IGNORE
538	<&>	IGNORE; IGNORE; IGNORE
539	<'>	IGNORE; IGNORE; IGNORE
540	<(>	IGNORE; IGNORE; IGNORE
541	<)>	IGNORE; IGNORE; IGNORE
542	<*>	IGNORE; IGNORE; IGNORE
543	<+>	IGNORE; IGNORE; IGNORE
544	<, >	IGNORE; IGNORE; IGNORE
545	<.>	IGNORE; IGNORE; IGNORE
546	<:>	IGNORE; IGNORE; IGNORE
547	<;>	IGNORE; IGNORE; IGNORE
548	<<>	IGNORE; IGNORE; IGNORE
549	<=>	IGNORE; IGNORE; IGNORE
550	</>>	IGNORE; IGNORE; IGNORE
551	<?>	IGNORE; IGNORE; IGNORE
552	<At>	IGNORE; IGNORE; IGNORE
553	<<(>	IGNORE; IGNORE; IGNORE
554	</ / / />	IGNORE; IGNORE; IGNORE
555	<)/>>	IGNORE; IGNORE; IGNORE
556	<' />>	IGNORE; IGNORE; IGNORE
557	<_>	IGNORE; IGNORE; IGNORE
558	<' !>	IGNORE; IGNORE; IGNORE
559	<(!>	IGNORE; IGNORE; IGNORE
560	<!!>	IGNORE; IGNORE; IGNORE
561	<!)>	IGNORE; IGNORE; IGNORE
562	<' ?>	IGNORE; IGNORE; IGNORE
563	<! I>	IGNORE; IGNORE; IGNORE
564	<Ct>	IGNORE; IGNORE; IGNORE
565	<Pd>	IGNORE; IGNORE; IGNORE
566	<Cu>	IGNORE; IGNORE; IGNORE
567	<Ye>	IGNORE; IGNORE; IGNORE
568	<BB>	IGNORE; IGNORE; IGNORE
569	<SE>	IGNORE; IGNORE; IGNORE
570	<' :>	IGNORE; IGNORE; IGNORE
571	<Co>	IGNORE; IGNORE; IGNORE
572	<-a>	IGNORE; IGNORE; IGNORE
573	<<<>	IGNORE; IGNORE; IGNORE
574	<NO>	IGNORE; IGNORE; IGNORE
575	<Rg>	IGNORE; IGNORE; IGNORE

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

```

576 <' ->      IGNORE; IGNORE; IGNORE
577 <DG>        IGNORE; IGNORE; IGNORE
578 <+ ->        IGNORE; IGNORE; IGNORE
579 <' ' >       IGNORE; IGNORE; IGNORE
580 <My>        IGNORE; IGNORE; IGNORE
581 <PI>        IGNORE; IGNORE; IGNORE
582 <.M>        IGNORE; IGNORE; IGNORE
583 <' , >       IGNORE; IGNORE; IGNORE
584 <-o>        IGNORE; IGNORE; IGNORE
585 </>/>>     IGNORE; IGNORE; IGNORE
586 <14>        IGNORE; IGNORE; IGNORE
587 <12>        IGNORE; IGNORE; IGNORE
588 <34>        IGNORE; IGNORE; IGNORE
589 <?I>        IGNORE; IGNORE; IGNORE
590 <*X>        IGNORE; IGNORE; IGNORE
591 <- : >       IGNORE; IGNORE; IGNORE
592 <' 6>       IGNORE; IGNORE; IGNORE
593 <" 6>       IGNORE; IGNORE; IGNORE
594 <- ! >      IGNORE; IGNORE; IGNORE
595 <-v>        IGNORE; IGNORE; IGNORE
596 <' 9>       IGNORE; IGNORE; IGNORE
597 <" 9>       IGNORE; IGNORE; IGNORE
598 <' 0>       IGNORE; IGNORE; IGNORE
599 <HB>        IGNORE; IGNORE; IGNORE
600 <TM>        IGNORE; IGNORE; IGNORE
601 <Md>        IGNORE; IGNORE; IGNORE
602 <18>        IGNORE; IGNORE; IGNORE
603 <38>        IGNORE; IGNORE; IGNORE
604 <58>        IGNORE; IGNORE; IGNORE
605 <78>        IGNORE; IGNORE; IGNORE
606 <Om>        IGNORE; IGNORE; IGNORE
607 <' ( >      IGNORE; IGNORE; IGNORE
608 <' ; >      IGNORE; IGNORE; IGNORE
609 <' <>       IGNORE; IGNORE; IGNORE
610 <' " >      IGNORE; IGNORE; IGNORE
611 <' . >      IGNORE; IGNORE; IGNORE
612 <;S>        IGNORE; IGNORE; IGNORE
613 <Vs>        IGNORE; IGNORE; IGNORE
614 <1M>        IGNORE; IGNORE; IGNORE
615 <1N>        IGNORE; IGNORE; IGNORE
616 <3M>        IGNORE; IGNORE; IGNORE
617 <4M>        IGNORE; IGNORE; IGNORE
618 <6M>        IGNORE; IGNORE; IGNORE
619 <1H>        IGNORE; IGNORE; IGNORE
620 <1T>        IGNORE; IGNORE; IGNORE
621 <-1>        IGNORE; IGNORE; IGNORE
622 <-N>        IGNORE; IGNORE; IGNORE
623 <-2>        IGNORE; IGNORE; IGNORE
624 <-M>        IGNORE; IGNORE; IGNORE
625 <-3>        IGNORE; IGNORE; IGNORE
626 <' 1>       IGNORE; IGNORE; IGNORE
627 <' 2>       IGNORE; IGNORE; IGNORE
628 <' 3>       IGNORE; IGNORE; IGNORE
629 <9' >       IGNORE; IGNORE; IGNORE
630 <9" >       IGNORE; IGNORE; IGNORE
631 <. 9>       IGNORE; IGNORE; IGNORE
632 <: 9>       IGNORE; IGNORE; IGNORE

```

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

633	<<1>	IGNORE; IGNORE; IGNORE
634	</>1>	IGNORE; IGNORE; IGNORE
635	<15>	IGNORE; IGNORE; IGNORE
636	<25>	IGNORE; IGNORE; IGNORE
637	<35>	IGNORE; IGNORE; IGNORE
638	<45>	IGNORE; IGNORE; IGNORE
639	<16>	IGNORE; IGNORE; IGNORE
640	<13>	IGNORE; IGNORE; IGNORE
641	<23>	IGNORE; IGNORE; IGNORE
642	<56>	IGNORE; IGNORE; IGNORE
643	<*->	IGNORE; IGNORE; IGNORE
644	</ /->	IGNORE; IGNORE; IGNORE
645	</ /=>	IGNORE; IGNORE; IGNORE
646	<-X>	IGNORE; IGNORE; IGNORE
647	<%0>	IGNORE; IGNORE; IGNORE
648	<co>	IGNORE; IGNORE; IGNORE
649	<PO>	IGNORE; IGNORE; IGNORE
650	<Rx>	IGNORE; IGNORE; IGNORE
651	<AO>	IGNORE; IGNORE; IGNORE
652	<oC>	IGNORE; IGNORE; IGNORE
653	<Ml>	IGNORE; IGNORE; IGNORE
654	<Fm>	IGNORE; IGNORE; IGNORE
655	<Tl>	IGNORE; IGNORE; IGNORE
656	<TR>	IGNORE; IGNORE; IGNORE
657	<MX>	IGNORE; IGNORE; IGNORE
658	<Mb>	IGNORE; IGNORE; IGNORE
659	<Mx>	IGNORE; IGNORE; IGNORE
660	<XX>	IGNORE; IGNORE; IGNORE
661	<OK>	IGNORE; IGNORE; IGNORE
662	<M2>	IGNORE; IGNORE; IGNORE
663	<! 2>	IGNORE; IGNORE; IGNORE
664	<=2>	IGNORE; IGNORE; IGNORE
665	<Ca>	IGNORE; IGNORE; IGNORE
666	<. .>	IGNORE; IGNORE; IGNORE
667	<. 3>	IGNORE; IGNORE; IGNORE
668	<: 3>	IGNORE; IGNORE; IGNORE
669	<. :>	IGNORE; IGNORE; IGNORE
670	<: .>	IGNORE; IGNORE; IGNORE
671	<-+>	IGNORE; IGNORE; IGNORE
672	<!=>	IGNORE; IGNORE; IGNORE
673	<=3>	IGNORE; IGNORE; IGNORE
674	<?1>	IGNORE; IGNORE; IGNORE
675	<?2>	IGNORE; IGNORE; IGNORE
676	<?->	IGNORE; IGNORE; IGNORE
677	<?=>	IGNORE; IGNORE; IGNORE
678	<=<>	IGNORE; IGNORE; IGNORE
679	</>=>	IGNORE; IGNORE; IGNORE
680	<0(>	IGNORE; IGNORE; IGNORE
681	<00>	IGNORE; IGNORE; IGNORE
682	<PP>	IGNORE; IGNORE; IGNORE
683	<-T>	IGNORE; IGNORE; IGNORE
684	<-L>	IGNORE; IGNORE; IGNORE
685	<-V>	IGNORE; IGNORE; IGNORE
686	<AN>	IGNORE; IGNORE; IGNORE
687	<OR>	IGNORE; IGNORE; IGNORE
688	<. P>	IGNORE; IGNORE; IGNORE
689	<dP>	IGNORE; IGNORE; IGNORE

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

```

690 <f (>      IGNORE; IGNORE; IGNORE
691 <In>        IGNORE; IGNORE; IGNORE
692 <Io>        IGNORE; IGNORE; IGNORE
693 <RT>        IGNORE; IGNORE; IGNORE
694 <*P>        IGNORE; IGNORE; IGNORE
695 <+Z>        IGNORE; IGNORE; IGNORE
696 <FA>        IGNORE; IGNORE; IGNORE
697 <TE>        IGNORE; IGNORE; IGNORE
698 <GF>        IGNORE; IGNORE; IGNORE
699 <DE>        IGNORE; IGNORE; IGNORE
700 <NB>        IGNORE; IGNORE; IGNORE
701 <( U>       IGNORE; IGNORE; IGNORE
702 <) U>       IGNORE; IGNORE; IGNORE
703 <( C>       IGNORE; IGNORE; IGNORE
704 <) C>       IGNORE; IGNORE; IGNORE
705 <( _>       IGNORE; IGNORE; IGNORE
706 <) _>       IGNORE; IGNORE; IGNORE
707 <( ->       IGNORE; IGNORE; IGNORE
708 <- )>       IGNORE; IGNORE; IGNORE
709 << />>     IGNORE; IGNORE; IGNORE
710 <UD>        IGNORE; IGNORE; IGNORE
711 <Ub>        IGNORE; IGNORE; IGNORE
712 <=>         IGNORE; IGNORE; IGNORE
713 <= />>     IGNORE; IGNORE; IGNORE
714 <==>       IGNORE; IGNORE; IGNORE
715 </ / 0>     IGNORE; IGNORE; IGNORE
716 <OL>        IGNORE; IGNORE; IGNORE
717 <0u>        IGNORE; IGNORE; IGNORE
718 <0U>        IGNORE; IGNORE; IGNORE
719 <SU>        IGNORE; IGNORE; IGNORE
720 <0 :>       IGNORE; IGNORE; IGNORE
721 <OS>        IGNORE; IGNORE; IGNORE
722 <fS>        IGNORE; IGNORE; IGNORE
723 <Or>        IGNORE; IGNORE; IGNORE
724 <SR>        IGNORE; IGNORE; IGNORE
725 <uT>        IGNORE; IGNORE; IGNORE
726 <UT>        IGNORE; IGNORE; IGNORE
727 <dT>        IGNORE; IGNORE; IGNORE
728 <Dt>        IGNORE; IGNORE; IGNORE
729 <PL>        IGNORE; IGNORE; IGNORE
730 <PR>        IGNORE; IGNORE; IGNORE
731 <*1>        IGNORE; IGNORE; IGNORE
732 <*2>        IGNORE; IGNORE; IGNORE
733 <VV>        IGNORE; IGNORE; IGNORE
734 <HH>        IGNORE; IGNORE; IGNORE
735 <DR>        IGNORE; IGNORE; IGNORE
736 <LD>        IGNORE; IGNORE; IGNORE
737 <UR>        IGNORE; IGNORE; IGNORE
738 <UL>        IGNORE; IGNORE; IGNORE
739 <VR>        IGNORE; IGNORE; IGNORE
740 <VL>        IGNORE; IGNORE; IGNORE
741 <DH>        IGNORE; IGNORE; IGNORE
742 <UH>        IGNORE; IGNORE; IGNORE
743 <VH>        IGNORE; IGNORE; IGNORE
744 <TB>        IGNORE; IGNORE; IGNORE
745 <LB>        IGNORE; IGNORE; IGNORE
746 <FB>        IGNORE; IGNORE; IGNORE

```

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

747	<sB>	IGNORE; IGNORE; IGNORE
748	<EH>	IGNORE; IGNORE; IGNORE
749	<vv>	IGNORE; IGNORE; IGNORE
750	<hh>	IGNORE; IGNORE; IGNORE
751	<dr>	IGNORE; IGNORE; IGNORE
752	<dl>	IGNORE; IGNORE; IGNORE
753	<ur>	IGNORE; IGNORE; IGNORE
754		IGNORE; IGNORE; IGNORE
755	<vr>	IGNORE; IGNORE; IGNORE
756	<vl>	IGNORE; IGNORE; IGNORE
757	<dh>	IGNORE; IGNORE; IGNORE
758	<uh>	IGNORE; IGNORE; IGNORE
759	<vh>	IGNORE; IGNORE; IGNORE
760	<.S>	IGNORE; IGNORE; IGNORE
761	<:S>	IGNORE; IGNORE; IGNORE
762	<?S>	IGNORE; IGNORE; IGNORE
763	<lB>	IGNORE; IGNORE; IGNORE
764	<RB>	IGNORE; IGNORE; IGNORE
765	<cC>	IGNORE; IGNORE; IGNORE
766	<cD>	IGNORE; IGNORE; IGNORE
767	<Dr>	IGNORE; IGNORE; IGNORE
768	<Dl>	IGNORE; IGNORE; IGNORE
769	<Ur>	IGNORE; IGNORE; IGNORE
770		IGNORE; IGNORE; IGNORE
771	<Vr>	IGNORE; IGNORE; IGNORE
772	<Vl>	IGNORE; IGNORE; IGNORE
773	<dH>	IGNORE; IGNORE; IGNORE
774	<uH>	IGNORE; IGNORE; IGNORE
775	<vH>	IGNORE; IGNORE; IGNORE
776	<Ob>	IGNORE; IGNORE; IGNORE
777	<Sb>	IGNORE; IGNORE; IGNORE
778	<Sn>	IGNORE; IGNORE; IGNORE
779	<Pt>	IGNORE; IGNORE; IGNORE
780	<NI>	IGNORE; IGNORE; IGNORE
781	<cH>	IGNORE; IGNORE; IGNORE
782	<cS>	IGNORE; IGNORE; IGNORE
783	<dR>	IGNORE; IGNORE; IGNORE
784	<dL>	IGNORE; IGNORE; IGNORE
785	<uR>	IGNORE; IGNORE; IGNORE
786		IGNORE; IGNORE; IGNORE
787	<vR>	IGNORE; IGNORE; IGNORE
788	<vL>	IGNORE; IGNORE; IGNORE
789	<Dh>	IGNORE; IGNORE; IGNORE
790	<Uh>	IGNORE; IGNORE; IGNORE
791	<Vh>	IGNORE; IGNORE; IGNORE
792	<Om>	IGNORE; IGNORE; IGNORE
793	<OM>	IGNORE; IGNORE; IGNORE
794	<Ic>	IGNORE; IGNORE; IGNORE
795	<SM>	IGNORE; IGNORE; IGNORE
796	<CG>	IGNORE; IGNORE; IGNORE
797	<Ci>	IGNORE; IGNORE; IGNORE
798	<(A>	IGNORE; IGNORE; IGNORE
799	</>V>	IGNORE; IGNORE; IGNORE
800	<!<>	IGNORE; IGNORE; IGNORE
801	<<*>	IGNORE; IGNORE; IGNORE
802	<! />>	IGNORE; IGNORE; IGNORE
803	<*/>>	IGNORE; IGNORE; IGNORE

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

```

804 <<7>      IGNORE; IGNORE; IGNORE
805 <7<>      IGNORE; IGNORE; IGNORE
806 </>7>    IGNORE; IGNORE; IGNORE
807 <7/>>    IGNORE; IGNORE; IGNORE
808 <I2>      IGNORE; IGNORE; IGNORE
809 <0.>      IGNORE; IGNORE; IGNORE
810 <HI>      IGNORE; IGNORE; IGNORE
811 <::>      IGNORE; IGNORE; IGNORE
812 <FD>      IGNORE; IGNORE; IGNORE
813 <LZ>      IGNORE; IGNORE; IGNORE
814 <BD>      IGNORE; IGNORE; IGNORE
815 <1R>      IGNORE; IGNORE; IGNORE
816 <2R>      IGNORE; IGNORE; IGNORE
817 <3R>      IGNORE; IGNORE; IGNORE
818 <4R>      IGNORE; IGNORE; IGNORE
819 <5R>      IGNORE; IGNORE; IGNORE
820 <6R>      IGNORE; IGNORE; IGNORE
821 <7R>      IGNORE; IGNORE; IGNORE
822 <8R>      IGNORE; IGNORE; IGNORE
823 <9R>      IGNORE; IGNORE; IGNORE
824 <aR>      IGNORE; IGNORE; IGNORE
825 <bR>      IGNORE; IGNORE; IGNORE
826 <cR>      IGNORE; IGNORE; IGNORE
827 <N0>      IGNORE; IGNORE; IGNORE
828 <i3>      IGNORE; IGNORE; IGNORE
829 <;>       IGNORE; IGNORE; IGNORE
830 <,>       IGNORE; IGNORE; IGNORE
831 <!*>      IGNORE; IGNORE; IGNORE
832 <?*>      IGNORE; IGNORE; IGNORE
833 <;'>      IGNORE; IGNORE; IGNORE
834 <,'>      IGNORE; IGNORE; IGNORE
835 <;!>      IGNORE; IGNORE; IGNORE
836 <,!>      IGNORE; IGNORE; IGNORE
837 <?;>      IGNORE; IGNORE; IGNORE
838 <?,>      IGNORE; IGNORE; IGNORE
839 <!:>      IGNORE; IGNORE; IGNORE
840 <?:>      IGNORE; IGNORE; IGNORE
841 <' %>     IGNORE; IGNORE; IGNORE
842 <,+>      IGNORE; IGNORE; IGNORE
843 <;+>      IGNORE; IGNORE; IGNORE
844 <?+>      IGNORE; IGNORE; IGNORE
845 <++>      IGNORE; IGNORE; IGNORE
846 <:+>      IGNORE; IGNORE; IGNORE
847 <" +>     IGNORE; IGNORE; IGNORE
848 <=+>      IGNORE; IGNORE; IGNORE
849 </ / +>   IGNORE; IGNORE; IGNORE
850 <' +>     IGNORE; IGNORE; IGNORE
851 <1+>      IGNORE; IGNORE; IGNORE
852 <3+>      IGNORE; IGNORE; IGNORE
853 <0+>      IGNORE; IGNORE; IGNORE
854 <IS>      IGNORE; IGNORE; IGNORE
855 <,_>      IGNORE; IGNORE; IGNORE
856 <._>      IGNORE; IGNORE; IGNORE
857 <+">      IGNORE; IGNORE; IGNORE
858 <+ _>     IGNORE; IGNORE; IGNORE
859 <* _>     IGNORE; IGNORE; IGNORE
860 <; _>     IGNORE; IGNORE; IGNORE

```

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

861	<0_>	IGNORE; IGNORE; IGNORE
862	<<+>	IGNORE; IGNORE; IGNORE
863	</>+>	IGNORE; IGNORE; IGNORE
864	<<'>	IGNORE; IGNORE; IGNORE
865	</>'>	IGNORE; IGNORE; IGNORE
866	<<">	IGNORE; IGNORE; IGNORE
867	</>">	IGNORE; IGNORE; IGNORE
868	<(">	IGNORE; IGNORE; IGNORE
869	<)">	IGNORE; IGNORE; IGNORE
870	<=/ />	IGNORE; IGNORE; IGNORE
871	<=_>	IGNORE; IGNORE; IGNORE
872	<('>	IGNORE; IGNORE; IGNORE
873	<)'>	IGNORE; IGNORE; IGNORE
874	<KM>	IGNORE; IGNORE; IGNORE
875	<"5>	IGNORE; IGNORE; IGNORE
876	<05>	IGNORE; IGNORE; IGNORE
877	<*5>	IGNORE; IGNORE; IGNORE
878	<+5>	IGNORE; IGNORE; IGNORE
879	<-6>	IGNORE; IGNORE; IGNORE
880	<*6>	IGNORE; IGNORE; IGNORE
881	<+6>	IGNORE; IGNORE; IGNORE
882	<Iu>	IGNORE; IGNORE; IGNORE
883	<Il>	IGNORE; IGNORE; IGNORE
884	<NU>	IGNORE; IGNORE; IGNORE
885	<SH>	IGNORE; IGNORE; IGNORE
886	<SX>	IGNORE; IGNORE; IGNORE
887	<EX>	IGNORE; IGNORE; IGNORE
888	<ET>	IGNORE; IGNORE; IGNORE
889	<EQ>	IGNORE; IGNORE; IGNORE
890	<AK>	IGNORE; IGNORE; IGNORE
891	<BL>	IGNORE; IGNORE; IGNORE
892	<BS>	IGNORE; IGNORE; IGNORE
893	<SO>	IGNORE; IGNORE; IGNORE
894	<SI>	IGNORE; IGNORE; IGNORE
895	<DL>	IGNORE; IGNORE; IGNORE
896	<D1>	IGNORE; IGNORE; IGNORE
897	<D2>	IGNORE; IGNORE; IGNORE
898	<D3>	IGNORE; IGNORE; IGNORE
899	<D4>	IGNORE; IGNORE; IGNORE
900	<NK>	IGNORE; IGNORE; IGNORE
901	<SY>	IGNORE; IGNORE; IGNORE
902	<EB>	IGNORE; IGNORE; IGNORE
903	<CN>	IGNORE; IGNORE; IGNORE
904		IGNORE; IGNORE; IGNORE
905	<SB>	IGNORE; IGNORE; IGNORE
906	<EC>	IGNORE; IGNORE; IGNORE
907	<FS>	IGNORE; IGNORE; IGNORE
908	<GS>	IGNORE; IGNORE; IGNORE
909	<RS>	IGNORE; IGNORE; IGNORE
910	<US>	IGNORE; IGNORE; IGNORE
911	<DT>	IGNORE; IGNORE; IGNORE
912	<PA>	IGNORE; IGNORE; IGNORE
913	<HO>	IGNORE; IGNORE; IGNORE
914	<BH>	IGNORE; IGNORE; IGNORE
915	<NH>	IGNORE; IGNORE; IGNORE
916	<IN>	IGNORE; IGNORE; IGNORE
917	<NL>	IGNORE; IGNORE; IGNORE

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

```

918 <SA>      IGNORE; IGNORE; IGNORE
919 <ES>      IGNORE; IGNORE; IGNORE
920 <HS>      IGNORE; IGNORE; IGNORE
921 <HJ>      IGNORE; IGNORE; IGNORE
922 <VS>      IGNORE; IGNORE; IGNORE
923 <PD>      IGNORE; IGNORE; IGNORE
924 <PU>      IGNORE; IGNORE; IGNORE
925 <RI>      IGNORE; IGNORE; IGNORE
926 <S2>      IGNORE; IGNORE; IGNORE
927 <S3>      IGNORE; IGNORE; IGNORE
928 <DC>      IGNORE; IGNORE; IGNORE
929 <P1>      IGNORE; IGNORE; IGNORE
930 <P2>      IGNORE; IGNORE; IGNORE
931 <TS>      IGNORE; IGNORE; IGNORE
932 <CC>      IGNORE; IGNORE; IGNORE
933 <MW>      IGNORE; IGNORE; IGNORE
934 <SG>      IGNORE; IGNORE; IGNORE
935 <EG>      IGNORE; IGNORE; IGNORE
936 <SS>      IGNORE; IGNORE; IGNORE
937 <GC>      IGNORE; IGNORE; IGNORE
938 <SC>      IGNORE; IGNORE; IGNORE
939 <CI>      IGNORE; IGNORE; IGNORE
940 <ST>      IGNORE; IGNORE; IGNORE
941 <OC>      IGNORE; IGNORE; IGNORE
942 <PM>      IGNORE; IGNORE; IGNORE
943 <AC>      IGNORE; IGNORE; IGNORE
944 <__>      IGNORE; IGNORE; IGNORE
945 <" !>      IGNORE; IGNORE; IGNORE
946 <" '>      IGNORE; IGNORE; IGNORE
947 <" />>      IGNORE; IGNORE; IGNORE
948 <" ?>      IGNORE; IGNORE; IGNORE
949 <" ->      IGNORE; IGNORE; IGNORE
950 <" (>      IGNORE; IGNORE; IGNORE
951 <" .>      IGNORE; IGNORE; IGNORE
952 <" :>      IGNORE; IGNORE; IGNORE
953 <" //>      IGNORE; IGNORE; IGNORE
954 <" 0>      IGNORE; IGNORE; IGNORE
955 <" ,>      IGNORE; IGNORE; IGNORE
956 <" _>      IGNORE; IGNORE; IGNORE
957 <" ">      IGNORE; IGNORE; IGNORE
958 <" <>      IGNORE; IGNORE; IGNORE
959 <" ;>      IGNORE; IGNORE; IGNORE
960 <" =>      IGNORE; IGNORE; IGNORE
961 <" 1>      IGNORE; IGNORE; IGNORE
962 <" 2>      IGNORE; IGNORE; IGNORE
963 <Fd>      IGNORE; IGNORE; IGNORE
964 <Bd>      IGNORE; IGNORE; IGNORE
965 <Fl>      IGNORE; IGNORE; IGNORE
966 <Li>      IGNORE; IGNORE; IGNORE
967 < //f>      IGNORE; IGNORE; IGNORE
968 <0s>      IGNORE; IGNORE; IGNORE
969 <1s>      IGNORE; IGNORE; IGNORE
970 <2s>      IGNORE; IGNORE; IGNORE
971 <3s>      IGNORE; IGNORE; IGNORE
972 <4s>      IGNORE; IGNORE; IGNORE
973 <5s>      IGNORE; IGNORE; IGNORE
974 <6s>      IGNORE; IGNORE; IGNORE

```

1

975	<7s>	IGNORE; IGNORE; IGNORE
976	<8s>	IGNORE; IGNORE; IGNORE
977	<9s>	IGNORE; IGNORE; IGNORE
978	<0S>	IGNORE; IGNORE; IGNORE
979	<4S>	IGNORE; IGNORE; IGNORE
980	<5S>	IGNORE; IGNORE; IGNORE
981	<6S>	IGNORE; IGNORE; IGNORE
982	<7S>	IGNORE; IGNORE; IGNORE
983	<8S>	IGNORE; IGNORE; IGNORE
984	<9S>	IGNORE; IGNORE; IGNORE
985	<+S>	IGNORE; IGNORE; IGNORE
986	<-S>	IGNORE; IGNORE; IGNORE
987	<1h>	IGNORE; IGNORE; IGNORE
988	<2h>	IGNORE; IGNORE; IGNORE
989	<3h>	IGNORE; IGNORE; IGNORE
990	<4h>	IGNORE; IGNORE; IGNORE
991	<1j>	IGNORE; IGNORE; IGNORE
992	<2j>	IGNORE; IGNORE; IGNORE
993	<3j>	IGNORE; IGNORE; IGNORE
994	<4j>	IGNORE; IGNORE; IGNORE
995	<UA>	IGNORE; IGNORE; IGNORE
996	<UB>	IGNORE; IGNORE; IGNORE
997	<yr>	IGNORE; IGNORE; IGNORE
998	<.6>	IGNORE; IGNORE; IGNORE
999	<<6>	IGNORE; IGNORE; IGNORE
1000	</>6>	IGNORE; IGNORE; IGNORE
1001	<,6>	IGNORE; IGNORE; IGNORE
1002	<&6>	IGNORE; IGNORE; IGNORE
1003	<(S>	IGNORE; IGNORE; IGNORE
1004	<)S>	IGNORE; IGNORE; IGNORE
1005	<UNDEFINED>	IGNORE; IGNORE; IGNORE
1006	<0>	
1007	<1>	<1>
1008	<1S>	<1>
1009	<2>	<2>
1010	<2S>	<2>
1011	<3>	<3>
1012	<3S>	<3>
1013	<4>	
1014	<5>	
1015	<6>	
1016	<7>	
1017	<8>	
1018	<9>	
1019	<A>	<A>; <NO-ACCENT>; <CAPITAL>
1020	<a>	<A>; <NO-ACCENT>; <SMALL>
1021	<A'>	<A>; <ACUTE>; <CAPITAL>
1022	<a'>	<A>; <ACUTE>; <SMALL>
1023	<A!>	<A>; <GRAVE>; <CAPITAL>
1024	<a!>	<A>; <GRAVE>; <SMALL>
1025	<A/>>	<A>; <CIRCUMFLEX>; <CAPITAL>
1026	<a/>>	<A>; <CIRCUMFLEX>; <SMALL>
1027	<A?>	<A>; <TILDE>; <CAPITAL>
1028	<a?>	<A>; <TILDE>; <SMALL>
1029	<A->	<A>; <MACRON>; <CAPITAL>
1030	<a->	<A>; <MACRON>; <SMALL>
1031	<A(>	<A>; <BREVE>; <CAPITAL>

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

1032	<a (>	<A>;<BREVE>;<SMALL>
1033	<A_>	<A>;<UNDERLINE>;<CAPITAL>
1034	<a_>	<A>;<UNDERLINE>;<SMALL>
1035	<A; >	<A>;<OGONEK>;<CAPITAL>
1036	<a; >	<A>;<OGONEK>;<SMALL>
1037	<A<>	<A>;<CARON>;<CAPITAL>
1038	<a<>	<A>;<CARON>;<SMALL>
1039	<'A>	<A>;<PRECEDED-BY-APOSTROPHE>;<CAPITAL>
1040	<'a>	<A>;<PRECEDED-BY-APOSTROPHE>;<SMALL>
1041	<A1>	<A>;<ACC1>;<CAPITAL>
1042	<a1>	<A>;<ACC1>;<SMALL>
1043	<A2>	<A>;<ACC2>;<CAPITAL>
1044	<a2>	<A>;<ACC2>;<SMALL>
1045		;<NO-ACCENT>;<CAPITAL>
1046		;<NO-ACCENT>;<SMALL>
1047	<B.>	;<DOT>;<CAPITAL>
1048	<b.>	;<DOT>;<SMALL>
1049	<B_>	;<UNDERLINE>;<CAPITAL>
1050	<b_>	;<UNDERLINE>;<SMALL>
1051	<'B>	;<PRECEDED-BY-APOSTROPHE>;<CAPITAL>
1052	<'b>	;<PRECEDED-BY-APOSTROPHE>;<SMALL>
1053	<C>	<C>;<NO-ACCENT>;<CAPITAL>
1054	<c>	<C>;<NO-ACCENT>;<SMALL>
1055	<C'>	<C>;<ACUTE>;<CAPITAL>
1056	<c'>	<C>;<ACUTE>;<SMALL>
1057	<C/>>	<C>;<CIRCUMFLEX>;<CAPITAL>
1058	<c/>>	<C>;<CIRCUMFLEX>;<SMALL>
1059	<C.>	<C>;<DOT>;<CAPITAL>
1060	<c.>	<C>;<DOT>;<SMALL>
1061	<C,>	<C>;<CEDILLA>;<CAPITAL>
1062	<c,>	<C>;<CEDILLA>;<SMALL>
1063	<C<>	<C>;<CARON>;<CAPITAL>
1064	<c<>	<C>;<CARON>;<SMALL>
1065	<D>	<D>;<NO-ACCENT>;<CAPITAL>
1066	<d>	<D>;<NO-ACCENT>;<SMALL>
1067	<D.>	<D>;<DOT>;<CAPITAL>
1068	<d.>	<D>;<DOT>;<SMALL>
1069	<D_>	<D>;<UNDERLINE>;<CAPITAL>
1070	<d_>	<D>;<UNDERLINE>;<SMALL>
1071	<D//>	<D>;<STROKE>;<CAPITAL>
1072	<d//>	<D>;<STROKE>;<SMALL>
1073	<D; >	<D>;<OGONEK>;<CAPITAL>
1074	<d; >	<D>;<OGONEK>;<SMALL>
1075	<D<>	<D>;<CARON>;<CAPITAL>
1076	<d<>	<D>;<CARON>;<SMALL>
1077	<'D>	<D>;<PRECEDED-BY-APOSTROPHE>;<CAPITAL>
1078	<'d>	<D>;<PRECEDED-BY-APOSTROPHE>;<SMALL>
1079	<D->	<D>;<SPECIAL>;<CAPITAL>
1080	<d->	<D>;<SPECIAL>;<SMALL>
1081	<E>	<E>;<NO-ACCENT>;<CAPITAL>
1082	<e>	<E>;<NO-ACCENT>;<SMALL>
1083	<E'>	<E>;<ACUTE>;<CAPITAL>
1084	<e'>	<E>;<ACUTE>;<SMALL>
1085	<E!>	<E>;<GRAVE>;<CAPITAL>
1086	<e!>	<E>;<GRAVE>;<SMALL>
1087	<E/>>	<E>;<CIRCUMFLEX>;<CAPITAL>
1088	<e/>>	<E>;<CIRCUMFLEX>;<SMALL>

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1089	<E?>	<E>;<TILDE>;<CAPITAL>
1090	<e?>	<E>;<TILDE>;<SMALL>
1091	<E->	<E>;<MACRON>;<CAPITAL>
1092	<e->	<E>;<MACRON>;<SMALL>
1093	<E(>	<E>;<BREVE>;<CAPITAL>
1094	<e(>	<E>;<BREVE>;<SMALL>
1095	<E.>	<E>;<DOT>;<CAPITAL>
1096	<e.>	<E>;<DOT>;<SMALL>
1097	<E:>	<E>;<DIAERESIS>;<CAPITAL>
1098	<e:>	<E>;<DIAERESIS>;<SMALL>
1099	<E_>	<E>;<UNDERLINE>;<CAPITAL>
1100	<e_>	<E>;<UNDERLINE>;<SMALL>
1101	<E;>	<E>;<OGONEK>;<CAPITAL>
1102	<e;>	<E>;<OGONEK>;<SMALL>
1103	<E<>	<E>;<CARON>;<CAPITAL>
1104	<e<>	<E>;<CARON>;<SMALL>
1105	<F>	<F>;<NO-ACCENT>;<CAPITAL>
1106	<f>	<F>;<NO-ACCENT>;<SMALL>
1107	<F.>	<F>;<DOT>;<CAPITAL>
1108	<f.>	<F>;<DOT>;<SMALL>
1109	<ff>	<FF+>;<SPECIAL>;<SMALL>
1110	<fi>	<FI+>;<SPECIAL>;<SMALL>
1111	<fl>	<FL+>;<SPECIAL>;<SMALL>
1112	<ft>	<FT+>;<SPECIAL>;<SMALL>
1113	<G>	<G>;<NO-ACCENT>;<CAPITAL>
1114	<g>	<G>;<NO-ACCENT>;<SMALL>
1115	<G'>	<G>;<ACUTE>;<CAPITAL>
1116	<g'>	<G>;<ACUTE>;<SMALL>
1117	<G/>>	<G>;<CIRCUMFLEX>;<CAPITAL>
1118	<g/>>	<G>;<CIRCUMFLEX>;<SMALL>
1119	<G->	<G>;<MACRON>;<CAPITAL>
1120	<g->	<G>;<MACRON>;<SMALL>
1121	<G(>	<G>;<BREVE>;<CAPITAL>
1122	<g(>	<G>;<BREVE>;<SMALL>
1123	<G.>	<G>;<DOT>;<CAPITAL>
1124	<g.>	<G>;<DOT>;<SMALL>
1125	<G,>	<G>;<CEDILLA>;<CAPITAL>
1126	<g,>	<G>;<CEDILLA>;<SMALL>
1127	<G//>	<G>;<STROKE>;<CAPITAL>
1128	<g//>	<G>;<STROKE>;<SMALL>
1129	<G<>	<G>;<CARON>;<CAPITAL>
1130	<g<>	<G>;<CARON>;<SMALL>
1131	<'G>	<G>;<PRECEDED-BY-APOSTROPHE>;<CAPITAL>
1132	<'g>	<G>;<PRECEDED-BY-APOSTROPHE>;<SMALL>
1133	<H>	<H>;<NO-ACCENT>;<CAPITAL>
1134	<h>	<H>;<NO-ACCENT>;<SMALL>
1135	<H/>>	<H>;<CIRCUMFLEX>;<CAPITAL>
1136	<h/>>	<H>;<CIRCUMFLEX>;<SMALL>
1137	<H.>	<H>;<DOT>;<CAPITAL>
1138	<h.>	<H>;<DOT>;<SMALL>
1139	<H:>	<H>;<DIAERESIS>;<CAPITAL>
1140	<h:>	<H>;<DIAERESIS>;<SMALL>
1141	<H,>	<H>;<CEDILLA>;<CAPITAL>
1142	<h,>	<H>;<CEDILLA>;<SMALL>
1143	<H//>	<H>;<STROKE>;<CAPITAL>
1144	<h//>	<H>;<STROKE>;<SMALL>
1145	<H;>	<H>;<OGONEK>;<CAPITAL>

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1146	<h;>	<H>;<OGONEK>;<SMALL>
1147	<I>	<I>;<NO-ACCENT>;<CAPITAL>
1148	<i>	<I>;<NO-ACCENT>;<SMALL>
1149	<I'>	<I>;<ACUTE>;<CAPITAL>
1150	<i'>	<I>;<ACUTE>;<SMALL>
1151	<I!>	<I>;<GRAVE>;<CAPITAL>
1152	<i!>	<I>;<GRAVE>;<SMALL>
1153	<I/>>	<I>;<CIRCUMFLEX>;<CAPITAL>
1154	<i/>>	<I>;<CIRCUMFLEX>;<SMALL>
1155	<I?>	<I>;<TILDE>;<CAPITAL>
1156	<i?>	<I>;<TILDE>;<SMALL>
1157	<I->	<I>;<MACRON>;<CAPITAL>
1158	<i->	<I>;<MACRON>;<SMALL>
1159	<I(>	<I>;<BREVE>;<CAPITAL>
1160	<i(>	<I>;<BREVE>;<SMALL>
1161	<I.>	<I>;<DOT>;<CAPITAL>
1162	<i.>	<I>;<DOT>;<SMALL>
1163	<I:>	<I>;<DIAERESIS>;<CAPITAL>
1164	<i:>	<I>;<DIAERESIS>;<SMALL>
1165	<I;>	<I>;<OGONEK>;<CAPITAL>
1166	<i;>	<I>;<OGONEK>;<SMALL>
1167	<I<>	<I>;<CARON>;<CAPITAL>
1168	<i<>	<I>;<CARON>;<SMALL>
1169	<I-J>	<I><J>;<I-J><I-J>;<CAPITAL><CAPITAL>
1170	<i-j>	<I><J>;<I-J><I-J>;<SMALL><SMALL>
1171	<IJ>	<I><J>;<IJ><IJ>;<CAPITAL><CAPITAL>
1172	<ij>	<I><J>;<IJ><IJ>;<SMALL><SMALL>
1173	<J>	<J>;<NO-ACCENT>;<CAPITAL>
1174	<j>	<J>;<NO-ACCENT>;<SMALL>
1175	<J/>>	<J>;<CIRCUMFLEX>;<CAPITAL>
1176	<j/>>	<J>;<CIRCUMFLEX>;<SMALL>
1177	<J(>	<J>;<BREVE>;<CAPITAL>
1178	<j(>	<J>;<BREVE>;<SMALL>
1179	<'J>	<J>;<PRECEDED-BY-APOSTROPHE>;<CAPITAL>
1180	<'j>	<J>;<PRECEDED-BY-APOSTROPHE>;<SMALL>
1181	<K>	<K>;<NO-ACCENT>;<CAPITAL>
1182	<k>	<K>;<NO-ACCENT>;<SMALL>
1183	<K'>	<K>;<ACUTE>;<CAPITAL>
1184	<k'>	<K>;<ACUTE>;<SMALL>
1185	<K.>	<K>;<DOT>;<CAPITAL>
1186	<k.>	<K>;<DOT>;<SMALL>
1187	<K,>	<K>;<CEDILLA>;<CAPITAL>
1188	<k,>	<K>;<CEDILLA>;<SMALL>
1189	<K_>	<K>;<UNDERLINE>;<CAPITAL>
1190	<k_>	<K>;<UNDERLINE>;<SMALL>
1191	<K;>	<K>;<OGONEK>;<CAPITAL>
1192	<k;>	<K>;<OGONEK>;<SMALL>
1193	<K<>	<K>;<CARON>;<CAPITAL>
1194	<k<>	<K>;<CARON>;<SMALL>
1195	<L>	<L>;<NO-ACCENT>;<CAPITAL>
1196	<l>	<L>;<NO-ACCENT>;<SMALL>
1197	<L'>	<L>;<ACUTE>;<CAPITAL>
1198	<l'>	<L>;<ACUTE>;<SMALL>
1199	<L.>	<L>;<DOT>;<CAPITAL>
1200	<l.>	<L>;<DOT>;<SMALL>
1201	<L,>	<L>;<CEDILLA>;<CAPITAL>
1202	<l,>	<L>;<CEDILLA>;<SMALL>

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1203	<L_>	<L>;<UNDERLINE>;<CAPITAL>
1204	<l_>	<L>;<UNDERLINE>;<SMALL>
1205	<L//>	<L>;<STROKE>;<CAPITAL>
1206	<l//>	<L>;<STROKE>;<SMALL>
1207	<L<>	<L>;<CARON>;<CAPITAL>
1208	<l<>	<L>;<CARON>;<SMALL>
1209	<M>	<M>;<NO-ACCENT>;<CAPITAL>
1210	<m>	<M>;<NO-ACCENT>;<SMALL>
1211	<M'>	<M>;<ACUTE>;<CAPITAL>
1212	<m'>	<M>;<ACUTE>;<SMALL>
1213	<M.>	<M>;<DOT>;<CAPITAL>
1214	<m.>	<M>;<DOT>;<SMALL>
1215	<N>	<N>;<NO-ACCENT>;<CAPITAL>
1216	<n>	<N>;<NO-ACCENT>;<SMALL>
1217	<N'>	<N>;<ACUTE>;<CAPITAL>
1218	<n'>	<N>;<ACUTE>;<SMALL>
1219	<N?>	<N>;<TILDE>;<CAPITAL>
1220	<n?>	<N>;<TILDE>;<SMALL>
1221	<N.>	<N>;<DOT>;<CAPITAL>
1222	<n.>	<N>;<DOT>;<SMALL>
1223	<N,>	<N>;<CEDILLA>;<CAPITAL>
1224	<n,>	<N>;<CEDILLA>;<SMALL>
1225	<N_>	<N>;<UNDERLINE>;<CAPITAL>
1226	<n_>	<N>;<UNDERLINE>;<SMALL>
1227	<N<>	<N>;<CARON>;<CAPITAL>
1228	<n<>	<N>;<CARON>;<SMALL>
1229	<'n>	<N>;<PRECEDED-BY-APOSTROPHE>;<SMALL>
1230	<N-G>	<N><G>;<N-G><N-G>;<CAPITAL><CAPITAL>
1231	<n-g>	<N><G>;<N-G><N-G>;<SMALL><SMALL>
1232	<NG>	<N><G>;<NG><NG>;<CAPITAL><CAPITAL>
1233	<ng>	<N><G>;<NG><NG>;<SMALL><SMALL>
1234	<O>	<O>;<NO-ACCENT>;<CAPITAL>
1235	<o>	<O>;<NO-ACCENT>;<SMALL>
1236	<O'>	<O>;<ACUTE>;<CAPITAL>
1237	<o'>	<O>;<ACUTE>;<SMALL>
1238	<O!>	<O>;<GRAVE>;<CAPITAL>
1239	<o!>	<O>;<GRAVE>;<SMALL>
1240	<O/>>	<O>;<CIRCUMFLEX>;<CAPITAL>
1241	<o/>>	<O>;<CIRCUMFLEX>;<SMALL>
1242	<O?>	<O>;<TILDE>;<CAPITAL>
1243	<o?>	<O>;<TILDE>;<SMALL>
1244	<O->	<O>;<MACRON>;<CAPITAL>
1245	<o->	<O>;<MACRON>;<SMALL>
1246	<O(>	<O>;<BREVE>;<CAPITAL>
1247	<o(>	<O>;<BREVE>;<SMALL>
1248	<O_>	<O>;<UNDERLINE>;<CAPITAL>
1249	<o_>	<O>;<UNDERLINE>;<SMALL>
1250	<O; >	<O>;<OGONEK>;<CAPITAL>
1251	<o; >	<O>;<OGONEK>;<SMALL>
1252	<O<>	<O>;<CARON>;<CAPITAL>
1253	<o<>	<O>;<CARON>;<SMALL>
1254	<O1>	<O>;<ACC1>;<CAPITAL>
1255	<o1>	<O>;<ACC1>;<SMALL>
1256	<O-E>	<O><E>;<O-E><O-E>;<CAPITAL><CAPITAL>
1257	<o-e>	<O><E>;<O-E><O-E>;<SMALL><SMALL>
1258	<OE>	<O><E>;<OE><OE>;<CAPITAL><CAPITAL>
1259	<oe>	<O><E>;<OE><OE>;<SMALL><SMALL>

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1260	<P>	<P>;<NO-ACCENT>;<CAPITAL>
1261	<p>	<P>;<NO-ACCENT>;<SMALL>
1262	<P'>	<P>;<ACUTE>;<CAPITAL>
1263	<p'>	<P>;<ACUTE>;<SMALL>
1264	<Q>	<Q>;<NO-ACCENT>;<CAPITAL>
1265	<q>	<Q>;<NO-ACCENT>;<SMALL>
1266	<kk>	<Q>;<SPECIAL>;<SMALL>
1267	<R>	<R>;<NO-ACCENT>;<CAPITAL>
1268	<r>	<R>;<NO-ACCENT>;<SMALL>
1269	<R'>	<R>;<ACUTE>;<CAPITAL>
1270	<r'>	<R>;<ACUTE>;<SMALL>
1271	<R.>	<R>;<DOT>;<CAPITAL>
1272	<r.>	<R>;<DOT>;<SMALL>
1273	<R,>	<R>;<CEDILLA>;<CAPITAL>
1274	<r,>	<R>;<CEDILLA>;<SMALL>
1275	<R_>	<R>;<UNDERLINE>;<CAPITAL>
1276	<r_>	<R>;<UNDERLINE>;<SMALL>
1277	<R<>	<R>;<CARON>;<CAPITAL>
1278	<r<>	<R>;<CARON>;<SMALL>
1279	<S>	<S>;<NO-ACCENT>;<CAPITAL>
1280	<s>	<S>;<NO-ACCENT>;<SMALL>
1281	<S'>	<S>;<ACUTE>;<CAPITAL>
1282	<s'>	<S>;<ACUTE>;<SMALL>
1283	<S/>>	<S>;<CIRCUMFLEX>;<CAPITAL>
1284	<s/>>	<S>;<CIRCUMFLEX>;<SMALL>
1285	<S.>	<S>;<DOT>;<CAPITAL>
1286	<s.>	<S>;<DOT>;<SMALL>
1287	<S,>	<S>;<CEDILLA>;<CAPITAL>
1288	<s,>	<S>;<CEDILLA>;<SMALL>
1289	<S;>	<S>;<OGONEK>;<CAPITAL>
1290	<s;>	<S>;<OGONEK>;<SMALL>
1291	<S<>	<S>;<CARON>;<CAPITAL>
1292	<s<>	<S>;<CARON>;<SMALL>
1293	<ss>	<S><S>;<ss><ss>;<SMALL><SMALL>
1294	<s-s>	<S><S>;<s-s><s-s>;<SMALL><SMALL>
1295	<st>	<ST+>;<SPECIAL>;<SMALL>
1296	<T>	<T>;<NO-ACCENT>;<CAPITAL>
1297	<t>	<T>;<NO-ACCENT>;<SMALL>
1298	<T.>	<T>;<DOT>;<CAPITAL>
1299	<t.>	<T>;<DOT>;<SMALL>
1300	<T,>	<T>;<CEDILLA>;<CAPITAL>
1301	<t,>	<T>;<CEDILLA>;<SMALL>
1302	<T_>	<T>;<UNDERLINE>;<CAPITAL>
1303	<t_>	<T>;<UNDERLINE>;<SMALL>
1304	<T//>	<T>;<STROKE>;<CAPITAL>
1305	<t//>	<T>;<STROKE>;<SMALL>
1306	<T<>	<T>;<CARON>;<CAPITAL>
1307	<t<>	<T>;<CARON>;<SMALL>
1308	<T-H>	<T><H>;<T-H><T-H>;<CAPITAL><CAPITAL>
1309	<t-h>	<T><H>;<T-H><T-H>;<SMALL><SMALL>
1310	<TH>	<T><H>;<TH><TH>;<CAPITAL><CAPITAL>
1311	<th>	<T><H>;<TH><TH>;<SMALL><SMALL>
1312	<U>	<U>;<NO-ACCENT>;<CAPITAL>
1313	<u>	<U>;<NO-ACCENT>;<SMALL>
1314	<U'>	<U>;<ACUTE>;<CAPITAL>
1315	<u'>	<U>;<ACUTE>;<SMALL>
1316	<U!>	<U>;<GRAVE>;<CAPITAL>

1
1

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

1317	<u!>	<U>;<GRAVE>;<SMALL>	
1318	<U/>>	<U>;<CIRCUMFLEX>;<CAPITAL>	
1319	<u/>>	<U>;<CIRCUMFLEX>;<SMALL>	
1320	<U?>	<U>;<TILDE>;<CAPITAL>	
1321	<u?>	<U>;<TILDE>;<SMALL>	
1322	<U->	<U>;<MACRON>;<CAPITAL>	
1323	<u->	<U>;<MACRON>;<SMALL>	
1324	<U(>	<U>;<BREVE>;<CAPITAL>	
1325	<u(>	<U>;<BREVE>;<SMALL>	
1326	<U; >	<U>;<OGONEK>;<CAPITAL>	1
1327	<u; >	<U>;<OGONEK>;<SMALL>	
1328	<U<>	<U>;<CARON>;<CAPITAL>	
1329	<u<>	<U>;<CARON>;<SMALL>	
1330	<U0>	<U>;<RING>;<CAPITAL>	1
1331	<u0>	<U>;<RING>;<SMALL>	1
1332	<V>	<V>;<NO-ACCENT>;<CAPITAL>	
1333	<v>	<V>;<NO-ACCENT>;<SMALL>	
1334	<V?>	<V>;<TILDE>;<CAPITAL>	
1335	<v?>	<V>;<TILDE>;<SMALL>	
1336	<W>	<W>;<NO-ACCENT>;<CAPITAL>	
1337	<w>	<W>;<NO-ACCENT>;<SMALL>	
1338	<W'>	<W>;<ACUTE>;<CAPITAL>	
1339	<w'>	<W>;<ACUTE>;<SMALL>	
1340	<W/>>	<W>;<CIRCUMFLEX>;<CAPITAL>	
1341	<w/>>	<W>;<CIRCUMFLEX>;<SMALL>	
1342	<W.>	<W>;<DOT>;<CAPITAL>	
1343	<w.>	<W>;<DOT>;<SMALL>	
1344	<W:>	<W>;<DIAERESIS>;<CAPITAL>	
1345	<w:>	<W>;<DIAERESIS>;<SMALL>	
1346	<X>	<X>;<NO-ACCENT>;<CAPITAL>	
1347	<x>	<X>;<NO-ACCENT>;<SMALL>	
1348	<X.>	<X>;<DOT>;<CAPITAL>	
1349	<x.>	<X>;<DOT>;<SMALL>	
1350	<X:>	<X>;<DIAERESIS>;<CAPITAL>	
1351	<x:>	<X>;<DIAERESIS>;<SMALL>	
1352	<Y>	<Y>;<NO-ACCENT>;<CAPITAL>	
1353	<y>	<Y>;<NO-ACCENT>;<SMALL>	
1354	<Y'>	<Y>;<ACUTE>;<CAPITAL>	
1355	<y'>	<Y>;<ACUTE>;<SMALL>	
1356	<Y!>	<Y>;<GRAVE>;<CAPITAL>	
1357	<y!>	<Y>;<GRAVE>;<SMALL>	
1358	<Y/>>	<Y>;<CIRCUMFLEX>;<CAPITAL>	
1359	<y/>>	<Y>;<CIRCUMFLEX>;<SMALL>	
1360	<Y.>	<Y>;<DOT>;<CAPITAL>	
1361	<y.>	<Y>;<DOT>;<SMALL>	
1362	<'Y>	<Y>;<PRECEDED-BY-APOSTROPHE>;<CAPITAL>	
1363	<'y>	<Y>;<PRECEDED-BY-APOSTROPHE>;<SMALL>	
1364	% <U:> and <U"> are treated as <Y> in Danish		1
1365	<U:>	<Y>;<ACC11>;<CAPITAL>	
1366	<u:>	<Y>;<ACC11>;<SMALL>	
1367	<U">	<Y>;<ACC12>;<CAPITAL>	
1368	<u">	<Y>;<ACC12>;<SMALL>	
1369	<Z>	<Z>;<NO-ACCENT>;<CAPITAL>	
1370	<z>	<Z>;<NO-ACCENT>;<SMALL>	
1371	<Z'>	<Z>;<ACUTE>;<CAPITAL>	
1372	<z'>	<Z>;<ACUTE>;<SMALL>	
1373	<Z/>>	<Z>;<CIRCUMFLEX>;<CAPITAL>	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1374	<z />	<Z>;<CIRCUMFLEX>;<SMALL>	
1375	<Z (>	<Z>;<BREVE>;<CAPITAL>	
1376	<z (>	<Z>;<BREVE>;<SMALL>	
1377	<Z .>	<Z>;<DOT>;<CAPITAL>	
1378	<z .>	<Z>;<DOT>;<SMALL>	
1379	<Z _>	<Z>;<UNDERLINE>;<CAPITAL>	
1380	<z _>	<Z>;<UNDERLINE>;<SMALL>	
1381	<Z //>	<Z>;<STROKE>;<CAPITAL>	
1382	<z //>	<Z>;<STROKE>;<SMALL>	
1383	<Z <>	<Z>;<CARON>;<CAPITAL>	
1384	<z <>	<Z>;<CARON>;<SMALL>	
1385	% <AE>	is treated as a separate letter in Danish	1
1386	<AE>	<AE>;<NO-ACCENT>;<CAPITAL>	
1387	<ae>	<AE>;<NO-ACCENT>;<SMALL>	
1388	<A :>	<AE>;<DIAERESIS>;<CAPITAL>	
1389	<a :>	<AE>;<DIAERESIS>;<SMALL>	
1390	<A 3>	<AE>;<ACC3>;<CAPITAL>	
1391	<a 3>	<AE>;<ACC3>;<SMALL>	
1392	% <O //>	is treated as a separate letter in Danish	1
1393	<O //>	<O //>;<NO-ACCENT>;<CAPITAL>	
1394	<o //>	<O //>;<NO-ACCENT>;<SMALL>	
1395	<O :>	<O //>;<DIAERESIS>;<CAPITAL>	
1396	<o :>	<O //>;<DIAERESIS>;<SMALL>	
1397	<O ">	<O //>;<DOUBLE-ACUTE>;<CAPITAL>	
1398	<o ">	<O //>;<DOUBLE-ACUTE>;<SMALL>	
1399	% <AA>	is treated as a separate letter in Danish	1
1400	<AA>	<AA>;<NO-ACCENT>;<CAPITAL>	
1401	<aa>	<AA>;<NO-ACCENT>;<SMALL>	
1402	<A -A>	<AA>;<ACC1>;<CAPITAL>	
1403	<A -a>	<AA>;<ACC1>;<BOTH>	
1404	<a -a>	<AA>;<ACC1>;<SMALL>	
1405	<A =>	<A =>;<CYRILLIC>;<CAPITAL>	
1406	<a =>	<A =>;<CYRILLIC>;<SMALL>	
1407	<B =>	<B =>;<CYRILLIC>;<CAPITAL>	
1408	<b =>	<B =>;<CYRILLIC>;<SMALL>	
1409	<V =>	<V =>;<CYRILLIC>;<CAPITAL>	
1410	<v =>	<V =>;<CYRILLIC>;<SMALL>	
1411	<G =>	<G =>;<CYRILLIC>;<CAPITAL>	
1412	<g =>	<G =>;<CYRILLIC>;<SMALL>	
1413	<G %>	<G =>;<ALPHA-1>;<CAPITAL>	
1414	<g %>	<G =>;<ALPHA-1>;<SMALL>	
1415	<D =>	<D =>;<CYRILLIC>;<CAPITAL>	
1416	<d =>	<D =>;<CYRILLIC>;<SMALL>	
1417	<D %>	<D %>;<ALPHA-1>;<CAPITAL>	
1418	<d %>	<D %>;<ALPHA-1>;<SMALL>	
1419	<E =>	<E =>;<CYRILLIC>;<CAPITAL>	
1420	<e =>	<E =>;<CYRILLIC>;<SMALL>	
1421	<IO>	<E =>;<SPECIAL>;<CAPITAL>	
1422	<io>	<E =>;<SPECIAL>;<SMALL>	
1423	<IE>	<IE>;<SPECIAL>;<CAPITAL>	
1424	<ie>	<IE>;<SPECIAL>;<SMALL>	
1425	<Z %>	<Z %>;<ALPHA-1>;<CAPITAL>	
1426	<z %>	<Z %>;<ALPHA-1>;<SMALL>	
1427	<Z =>	<Z =>;<CYRILLIC>;<CAPITAL>	
1428	<z =>	<Z =>;<CYRILLIC>;<SMALL>	
1429	<DS>	<DS>;<SPECIAL>;<CAPITAL>	1

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

1430	<ds>	<DS>; <SPECIAL>; <SMALL>
1431	<I=>	<I=>; <CYRILLIC>; <CAPITAL>
1432	<i=>	<I=>; <CYRILLIC>; <SMALL>
1433	<II>	<II>; <SPECIAL>; <CAPITAL>
1434	<ii>	<II>; <SPECIAL>; <SMALL>
1435	<YI>	<II>; <ALPHA-1>; <CAPITAL>
1436	<yi>	<II>; <ALPHA-1>; <SMALL>
1437	<J%>	<J%>; <ALPHA-1>; <CAPITAL>
1438	<j%>	<J%>; <ALPHA-1>; <SMALL>
1439	<J=>	<J=>; <CYRILLIC>; <CAPITAL>
1440	<j=>	<J=>; <CYRILLIC>; <SMALL>
1441	<K=>	<K=>; <CYRILLIC>; <CAPITAL>
1442	<k=>	<K=>; <CYRILLIC>; <SMALL>
1443	<KJ>	<K=>; <SPECIAL>; <CAPITAL>
1444	<kj>	<K=>; <SPECIAL>; <SMALL>
1445	<L=>	<L=>; <CYRILLIC>; <CAPITAL>
1446	<l=>	<L=>; <CYRILLIC>; <SMALL>
1447	<LJ>	<LJ>; <SPECIAL>; <CAPITAL>
1448	<l j>	<LJ>; <SPECIAL>; <SMALL>
1449	<M=>	<M=>; <CYRILLIC>; <CAPITAL>
1450	<m=>	<M=>; <CYRILLIC>; <SMALL>
1451	<N=>	<N=>; <CYRILLIC>; <CAPITAL>
1452	<n=>	<N=>; <CYRILLIC>; <SMALL>
1453	<NJ>	<NJ>; <SPECIAL>; <CAPITAL>
1454	<nj>	<NJ>; <SPECIAL>; <SMALL>
1455	<O=>	<O=>; <CYRILLIC>; <CAPITAL>
1456	<o=>	<O=>; <CYRILLIC>; <SMALL>
1457	<P=>	<P=>; <CYRILLIC>; <CAPITAL>
1458	<p=>	<P=>; <CYRILLIC>; <SMALL>
1459	<R=>	<R=>; <CYRILLIC>; <CAPITAL>
1460	<r=>	<R=>; <CYRILLIC>; <SMALL>
1461	<S=>	<S=>; <CYRILLIC>; <CAPITAL>
1462	<s=>	<S=>; <CYRILLIC>; <SMALL>
1463	<T=>	<T=>; <CYRILLIC>; <CAPITAL>
1464	<t=>	<T=>; <CYRILLIC>; <SMALL>
1465	<Ts>	<Ts>; <SPECIAL>; <CAPITAL>
1466	<ts>	<Ts>; <SPECIAL>; <SMALL>
1467	<U=>	<U=>; <CYRILLIC>; <CAPITAL>
1468	<u=>	<U=>; <CYRILLIC>; <SMALL>
1469	<V%>	<V%>; <ALPHA-1>; <CAPITAL>
1470	<v%>	<V%>; <ALPHA-1>; <SMALL>
1471	<F=>	<F=>; <CYRILLIC>; <CAPITAL>
1472	<f=>	<F=>; <CYRILLIC>; <SMALL>
1473	<H=>	<H=>; <CYRILLIC>; <CAPITAL>
1474	<h=>	<H=>; <CYRILLIC>; <SMALL>
1475	<C=>	<C=>; <CYRILLIC>; <CAPITAL>
1476	<c=>	<C=>; <CYRILLIC>; <SMALL>
1477	<C%>	<C%>; <ALPHA-1>; <CAPITAL>
1478	<c%>	<C%>; <ALPHA-1>; <SMALL>
1479	<DZ>	<DZ>; <SPECIAL>; <CAPITAL>
1480	<dz>	<DZ>; <SPECIAL>; <SMALL>
1481	<S%>	<S%>; <ALPHA-1>; <CAPITAL>
1482	<s%>	<S%>; <ALPHA-1>; <SMALL>
1483	<Sc>	<Sc>; <SPECIAL>; <CAPITAL>
1484	<sc>	<Sc>; <SPECIAL>; <SMALL>
1485	<='>	<='>; <ACUTE>; <SMALL>
1486	<=">	<=">; <DOUBLE-ACUTE>; <CAPITAL>

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1487	<Y=>	<Y=>; <CYRILLIC>; <CAPITAL>	
1488	<y=>	<Y=>; <CYRILLIC>; <SMALL>	
1489	<% ' >	<% ' >; <ACUTE>; <SMALL>	
1490	<% " >	<% ' >; <DOUBLE-ACUTE>; <CAPITAL>	
1491	<JE>	<JE>; <SPECIAL>; <CAPITAL>	
1492	<je>	<JE>; <SPECIAL>; <SMALL>	
1493	<JU>	<JU>; <SPECIAL>; <CAPITAL>	
1494	<ju>	<JU>; <SPECIAL>; <SMALL>	
1495	<JA>	<JA>; <SPECIAL>; <CAPITAL>	
1496	<ja>	<JA>; <SPECIAL>; <SMALL>	
1497	<A*>	<A*>; <GREEK>; <CAPITAL>	1
1498	<a*>	<A*>; <GREEK>; <SMALL>	
1499	<A%>	<A*>; <ALPHA-1>; <CAPITAL>	
1500	<a%>	<A*>; <ALPHA-1>; <SMALL>	
1501	<B*>	<B*>; <GREEK>; <CAPITAL>	
1502	<b*>	<B*>; <GREEK>; <SMALL>	
1503	<G*>	<G*>; <GREEK>; <CAPITAL>	
1504	<g*>	<G*>; <GREEK>; <SMALL>	
1505	<D*>	<D*>; <GREEK>; <CAPITAL>	
1506	<d*>	<D*>; <GREEK>; <SMALL>	
1507	<E*>	<E*>; <GREEK>; <CAPITAL>	
1508	<e*>	<E*>; <GREEK>; <SMALL>	
1509	<E%>	<E*>; <ALPHA-1>; <CAPITAL>	
1510	<e%>	<E*>; <ALPHA-1>; <SMALL>	
1511	<Z*>	<Z*>; <GREEK>; <CAPITAL>	
1512	<z*>	<Z*>; <GREEK>; <SMALL>	
1513	<Y*>	<Y*>; <GREEK>; <CAPITAL>	
1514	<y*>	<Y*>; <GREEK>; <SMALL>	
1515	<Y%>	<Y*>; <ALPHA-1>; <CAPITAL>	
1516	<y%>	<Y*>; <ALPHA-1>; <SMALL>	
1517	<H*>	<H*>; <GREEK>; <CAPITAL>	
1518	<h*>	<H*>; <GREEK>; <SMALL>	
1519	<I*>	<I*>; <GREEK>; <CAPITAL>	1
1520	<J*>	<I*>; <GREEK>; <CAPITAL>	1
1521	<i*>	<I*>; <GREEK>; <SMALL>	1
1522	<j*>	<I*>; <GREEK>; <SMALL>	1
1523	<I%>	<I*>; <ALPHA-1>; <CAPITAL>	1
1524	<i%>	<I*>; <ALPHA-1>; <SMALL>	1
1525	<K*>	<K*>; <GREEK>; <CAPITAL>	1
1526	<k*>	<K*>; <GREEK>; <SMALL>	1
1527	<L*>	<L*>; <GREEK>; <CAPITAL>	1
1528	<l*>	<L*>; <GREEK>; <SMALL>	1
1529	<M*>	<M*>; <GREEK>; <CAPITAL>	1
1530	<m*>	<M*>; <GREEK>; <SMALL>	1
1531	<N*>	<N*>; <GREEK>; <CAPITAL>	1
1532	<n*>	<N*>; <GREEK>; <SMALL>	1
1533	<C*>	<C*>; <GREEK>; <CAPITAL>	1
1534	<c*>	<C*>; <GREEK>; <SMALL>	1
1535	<O*>	<O*>; <GREEK>; <CAPITAL>	
1536	<o*>	<O*>; <GREEK>; <SMALL>	
1537	<O%>	<O*>; <ALPHA-1>; <CAPITAL>	
1538	<o%>	<O*>; <ALPHA-1>; <SMALL>	
1539	<P*>	<P*>; <GREEK>; <CAPITAL>	
1540	<p*>	<P*>; <GREEK>; <SMALL>	
1541	<R*>	<R*>; <GREEK>; <CAPITAL>	
1542	<r*>	<R*>; <GREEK>; <SMALL>	
1543	<S*>	<S*>; <GREEK>; <CAPITAL>	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

1544	<s*>	<S*>; <GREEK>; <SMALL>
1545	<*s>	<S*>; <SPECIAL>; <SMALL>
1546	<T*>	<T*>; <GREEK>; <CAPITAL>
1547	<t*>	<T*>; <GREEK>; <SMALL>
1548	<U*>	<U*>; <GREEK>; <CAPITAL>
1549	<V*>	<U*>; <GREEK>; <CAPITAL>
1550	<u*>	<U*>; <GREEK>; <SMALL>
1551	<v*>	<U*>; <GREEK>; <SMALL>
1552	<U%>	<U*>; <ALPHA-1>; <CAPITAL>
1553	<u%>	<U*>; <ALPHA-1>; <SMALL>
1554	<F*>	<F*>; <GREEK>; <CAPITAL>
1555	<f*>	<F*>; <GREEK>; <SMALL>
1556	<X*>	<X*>; <GREEK>; <CAPITAL>
1557	<x*>	<X*>; <GREEK>; <SMALL>
1558	<Q*>	<Q*>; <GREEK>; <CAPITAL>
1559	<q*>	<Q*>; <GREEK>; <SMALL>
1560	<W*>	<W*>; <GREEK>; <CAPITAL>
1561	<w*>	<W*>; <GREEK>; <SMALL>
1562	<W%>	<W*>; <ALPHA-1>; <CAPITAL>
1563	<w%>	<W*>; <ALPHA-1>; <SMALL>
1564	<p+>	
1565	<v+>	
1566	<gf>	
1567	<H'>	
1568	<aM>	
1569	<aH>	
1570	<wH>	
1571	<ah>	
1572	<yH>	
1573	<a+>	
1574	<b+>	
1575	<tm>	
1576	<t+>	
1577	<tk>	
1578	<g+>	
1579	<hk>	
1580	<x+>	
1581	<d+>	
1582	<dk>	
1583	<r+>	
1584	<z+>	
1585	<s+>	
1586	<sn>	
1587	<c+>	
1588	<dd>	
1589	<tj>	
1590	<zH>	
1591	<e+>	
1592	<i+>	
1593	<f+>	
1594	<q+>	
1595	<k+>	
1596	<l+>	
1597	<m+>	
1598	<n+>	
1599	<h+>	
1600	<w+>	

1601 <j+>
1602 <y+>
1603 <yf>
1604 <A+>
1605 <B+>
1606 <G+>
1607 <D+>
1608 <H+>
1609 <W+>
1610 <Z+>
1611 <X+>
1612 <Tj>
1613 <J+>
1614 <K%>
1615 <K+>
1616 <L+>
1617 <M%>
1618 <M+>
1619 <N%>
1620 <N+>
1621 <S+>
1622 <E+>
1623 <P%>
1624 <P+>
1625 <Zj>
1626 <ZJ>
1627 <Q+>
1628 <R+>
1629 <Sh>
1630 <T+>
1631 <b4>
1632 <p4>
1633 <m4>
1634 <f4>
1635 <d4>
1636 <t4>
1637 <n4>
1638 <l4>
1639 <g4>
1640 <k4>
1641 <h4>
1642 <j4>
1643 <q4>
1644 <x4>
1645 <zh>
1646 <ch>
1647 <sh>
1648 <r4>
1649 <z4>
1650 <c4>
1651 <s4>
1652 <a4>
1653 <o4>
1654 <e4>
1655 <eh>
1656 <ai>
1657 <ei>

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

1658 <au>
1659 <ou>
1660 <an>
1661 <en>
1662 <aN>
1663 <eN>
1664 <er>
1665 <i4>
1666 <u4>
1667 <iu>
1668 <A5>
1669 <a5>
1670 <I5>
1671 <i5>
1672 <U5>
1673 <u5>
1674 <E5>
1675 <e5>
1676 <O5>
1677 <o5>
1678 <ka>
1679 <ga>
1680 <ki>
1681 <gi>
1682 <ku>
1683 <gu>
1684 <ke>
1685 <ge>
1686 <ko>
1687 <go>
1688 <sa>
1689 <za>
1690 <si>
1691 <zi>
1692 <su>
1693 <zu>
1694 <se>
1695 <ze>
1696 <so>
1697 <zo>
1698 <ta>
1699 <da>
1700 <ti>
1701 <di>
1702 <tU>
1703 <tu>
1704 <du>
1705 <te>
1706 <de>
1707 <to>
1708 <do>
1709 <na>
1710 <ni>
1711 <nu>
1712 <ne>
1713 <no>
1714 <ha>

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

1715 <ba>
1716 <pa>
1717 <hi>
1718 <bi>
1719 <pi>
1720 <hu>
1721 <bu>
1722 <pu>
1723 <he>
1724 <be>
1725 <pe>
1726 <ho>
1727 <bo>
1728 <po>
1729 <ma>
1730 <mi>
1731 <mu>
1732 <me>
1733 <mo>
1734 <yA>
1735 <ya>
1736 <yU>
1737 <yu>
1738 <yO>
1739 <yo>
1740 <ra>
1741 <ri>
1742 <ru>
1743 <re>
1744 <ro>
1745 <wA>
1746 <wa>
1747 <wi>
1748 <we>
1749 <wo>
1750 <n5>
1751 <a6>
1752 <A6>
1753 <i6>
1754 <I6>
1755 <u6>
1756 <U6>
1757 <e6>
1758 <E6>
1759 <o6>
1760 <O6>
1761 <Ka>
1762 <Ga>
1763 <Ki>
1764 <Gi>
1765 <Ku>
1766 <Gu>
1767 <Ke>
1768 <Ge>
1769 <Ko>
1770 <Go>
1771 <Sa>

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

1772 <Za>
1773 <Si>
1774 <Zi>
1775 <Su>
1776 <Zu>
1777 <Se>
1778 <Ze>
1779 <So>
1780 <Zo>
1781 <Ta>
1782 <Da>
1783 <Ti>
1784 <Di>
1785 <TU>
1786 <Tu>
1787 <Du>
1788 <Te>
1789 <De>
1790 <To>
1791 <Do>
1792 <Na>
1793 <Ni>
1794 <Nu>
1795 <Ne>
1796 <No>
1797 <Ha>
1798 <Ba>
1799 <Pa>
1800 <Hi>
1801 <Bi>
1802 <Pi>
1803 <Hu>
1804 <Bu>
1805 <Pu>
1806 <He>
1807 <Be>
1808 <Pe>
1809 <Ho>
1810 <Bo>
1811 <Po>
1812 <Ma>
1813 <Mi>
1814 <Mu>
1815 <Me>
1816 <Mo>
1817 <YA>
1818 <Ya>
1819 <YU>
1820 <Yu>
1821 <YO>
1822 <Yo>
1823 <Ra>
1824 <Ri>
1825 <Ru>
1826 <Re>
1827 <Ro>
1828 <WA>

1
1

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```

1829 <Wa>
1830 <Wi>
1831 <We>
1832 <Wo>
1833 <N6>
1834 <Vu>
1835 <KA>
1836 <KE>

1837 order_end

1838 END LC_COLLATE

1839 LC_MONETARY

1840 % int_curr_symbol according to ISO 4217 1
1841 int_curr_symbol "DKK " 1
1842 currency_symbol "kr." 1
1843 mon_decimal_point <,>
1844 mon_thousands_sep <.>
1845 mon_grouping 3;0
1846 positive_sign ""
1847 negative_sign <->
1848 int_frac_digits 2
1849 frac_digits 2
1850 p_cs_precedes 1
1851 p_sep_by_space 1
1852 n_cs_precedes 1
1853 n_sep_by_space 1
1854 p_sign_posn 4
1855 n_sign_posn 4

1856 END LC_MONETARY

1857 LC_NUMERIC

1858 decimal_point <,>
1859 thousands_sep <.>
1860 grouping 3;0

1861 END LC_NUMERIC

1862 LC_TIME

1863 abday "s<o//>n";"man";"tir";"ons";"tor";"fre";"l<o//>r" 1
1864 day "s<o//>ndag";"mandag";"tirsdag";"onsdag";/ 1
1865 "torsdag";"fredag";"l<o//>rdag" 1
1866 abmon "jan";"feb";"mar";"apr";"maj";"jun";/ 1
1867 "jul";"aug";"sep";"okt";"nov";"dec" 1
1868 mon "januar";"februar";"marts";"april";"maj";"juni";/ 1
1869 "juli";"august";"september";"oktober";"november";"december" 1
1870 d_t_fmt "%a %d %b %Y %T %Z" 1
1871 d_fmt "%d %b %Y" 1
1872 t_fmt "%T" 1

1873 % The AM/PM notation is not used in Denmark and thus not allowed. 1
1874 am_pm "";" "

```

```

1875  t_fmt_ampm    " "
1876  END LC_TIME
1877  LC_MESSAGES

1878  % Must be careful to avoid interpreting "nej" (no) as "ja" (yes). 1
1879  % yesexpr      "^[[[:blank:]]*[JjYy]([[[:alpha:]]]*" 1
1880  % noexpr       "^[[[:blank:]]*[Nn]([[[:alpha:]]]*" 1

1881  yesexpr        "<'//><<(><<(>:blank:<)//><)//>*<<(>JjYy<)//>/ 1
1882  <<(><<(>:alpha:<)//><)//>*" 1
1883  noexpr         "<'//><<(><<(>:blank:<)//><)//>*<<(>Nn<)//>/ 1
1884  <<(><<(>:alpha:<)//><)//>*" 1

1885  END LC_MESSAGES

1886  F.3.2 fo_DK — (Example) Faroese LC_TIME and LC_MESSAGES

1887  escape_char    / 1
1888  comment_char   % 1
1889  % Danish example national locale for the Faroese language 1
1890  % Source: Danish Standards Association 1
1891  % Revision: 1.7 1991-04-26 1
1892  % 1
1893  % Only LC_TIME and LC_MESSAGES are specified here, else use the da_DK locale 1

1894  LC_CTYPE
1895  copy da_DK 1
1896  END LC_CTYPE

1897  LC_COLLATE 1
1898  copy da_DK 1
1899  END LC_COLLATE

1900  LC_MONETARY 1
1901  copy da_DK 1
1902  END LC_MONETARY

1903  LC_NUMERIC 1
1904  copy da_DK 1
1905  END LC_NUMERIC

1906  LC_TIME 1

1907  abday         "sun";"m<a'>n";"t<y'>s";"mik";"h<o'>s";"fr<i'>";"ley" 1
1908  day           "sunnudagur";"m<a'>nadagur";"t<y'>sdagur";/ 1
1909             "mikudagur";"h<o'>sdagur";"fr<i'>ggjadagur";"leygardagur" 1
1910  abmon         "jan";"feb";"mar";"apr";"mai";"jun";/ 1
1911             "jul";"aug";"sep";"okt";"nov";"des" 1
1912  mon           "januar";"februar";"mars";"apr<i'>l";"mai";"juni";/ 1
1913             "juli";"august";"september";"oktober";"november";"desember" 1
1914  d_t_fmt       "%a %d %b %Y %T %Z" 1
1915  d_fmt         "%d %b %Y" 1
1916  t_fmt         "%T" 1

```

```

1917 am_pm      "" ; ""
1918 t_fmt_ampm  ""

1919 END LC_TIME

1920 LC_MESSAGES

1921 % Must be careful to avoid interpreting "nej"/"nei" (no) as "ja" (yes).      1

1922 % yesexpr    "^[[[:blank:]]*[JjYy][[:alpha:]]*"                          1
1923 % noexpr     "^[[[:blank:]]*[Nn][[:alpha:]]*"                            1

1924 yesexpr      "<' />><<(><<(>:blank:<)/>><)/>>*<<(>JjYy<)/>>/"          1
1925 <<(><<(>:alpha:<)/>><)/>>*"                                                1
1926 noexpr       "<' />><<(><<(>:blank:<)/>><)/>>*<<(>Nn<)/>>/"            1
1927 <<(><<(>:alpha:<)/>><)/>>*"                                                1

1928 END LC_MESSAGES

1929 F.3.3 k1_DK — (Example) Greenlandic LC_TIME and LC_MESSAGES

1930 escape_char /                                                              1
1931 comment_char %                                                            1
1932 % Danish example national locale for the Greenlandic language            1
1933 % Source: Danish Standards Association                                    1
1934 % Revision: 1.7 1991-04-26                                                1
1935 %                                                                          1
1936 % Only LC_TIME and LC_MESSAGES are specified here, else use the da_DK locale 1

1937 LC_CTYPE
1938 copy da_DK                                                                1
1939 END LC_CTYPE

1940 LC_COLLATE
1941 copy da_DK                                                                1
1942 END LC_COLLATE

1943 LC_MONETARY
1944 copy da_DK                                                                1
1945 END LC_MONETARY

1946 LC_NUMERIC
1947 copy da_DK                                                                1
1948 END LC_NUMERIC

1949 LC_TIME                                                                    1

1950 abday      "sab";"ata";"mar";"pin";"sis";"tal";"arf"                    1
1951 day         "sabaat";"ataasinngorneq";"marlunngorneq";"pingasunngorneq";/ 1
1952           "sisamannngorneq";"tallimannngorneq";"arfininngorneq"          1
1953 abmon       "jan";"feb";"mar";"apr";"maj";"jun";/                        1
1954           "jul";"aug";"sep";"okt";"nov";"dec"                            1
1955 mon         "januari";"februari";"marts";"aprili";"maji";"juni";/        1
1956           "juli";"augustusi";"septemberi";"oktoberi";"novemberi";"decemberi" 1
1957 d_t_fmt     "%a %d %b %Y %T %Z"                                          1
1958 d_fmt       "%d %b %Y"                                                  1

```

```

1959  t_fmt          "%T"
1960  am_pm          " " ; " "
1961  t_fmt_ampm     " "

1962  END LC_TIME

1963  LC_MESSAGES

1964  % Must be careful to avoid interpreting "namik"/"nej" (no) as "aap"/"ja" (yes). 1
1965  % yesexpr       "^[[[:blank:]]*[JjYyAa][[:alpha:]]*" 1
1966  % noexpr        "^[[[:blank:]]*[Nn][[:alpha:]]*" 1

1967  yesexpr         "<' />><<(><<(>:blank:<)/>><)/>>*<<(>JjYyAa<)/>>/ 1
1968  <<(><<(>:alpha:<)/>><)/>>*" 1
1969  noexpr          "<' />><<(><<(>:blank:<)/>><)/>>*<<(>Nn<)/>>/ 1
1970  <<(><<(>:alpha:<)/>><)/>>*" 1

1971  END LC_MESSAGES

```

1972 F.4 Character Mnemonics Guidelines

1973 This clause presents guidelines for character mnemonics in a minimal coded char-
 1974 acter set. These guidelines are used within this sample annex and are recom-
 1975 mended for other national profiles.

1976 F.4.1 Aim of Character Mnemonics

1977 The aim of the mnemonics is to be able to represent all characters in all standard
 1978 coded character sets in any standard coded character set.

1979 The usage of the character mnemonics is primarily intended within computer
 1980 operating systems, programming languages, and applications and this work with
 1981 character mnemonics is the current state of work that has been presented to the
 1982 ISO working group responsible for these computer related issues, namely the
 1983 ISO/IEC JTC 1/SC22 special working group on coded character set usage.

1984 F.4.2 Covered Coded Character Sets

1985 All characters in the standard coded character sets will be given a mnemonic to be
 1986 represented in the minimal character set. The minimal coded character set is
 1987 defined as the basic character set of ISO 646 {1}, where 12 positions are left
 1988 undefined. The standard coded character sets are taken as the sum of all ISO-
 1989 defined or ISO-registered coded character sets.

1990 The most significant ISO coded character set is the ISO 10646 {B11} coded charac-
 1991 ter set, whose aim is to code in 32 bits all characters in the world. These guide-
 1992 lines can be seen as assigning mnemonic attributes to most characters in
 1993 ISO 10646 {B11}, currently at the DIS stage.

1994 Other ISO coded character sets covered include all parts of ISO 8859 {B9}
 1995 ISO 6937-2 {B6}, and all ISO 646 {1} conforming coded character sets in the ISO
 1996 character set registry managed by ECMA according to ISO 4873 {4}. Some non-ISO
 1997 coded character sets are also covered for convenience.

1998 **F.4.3 Character Mnemonics Classes**

1999 The character mnemonics are classified into two groups:

- 2000 (1) A group with two-character mnemonics—Primarily intended for alpha-
 2001 betic scripts like Latin, Greek, Cyrillic, Hebrew, and Arabic, and special
 2002 characters.
- 2003 (2) A group with variable-length mnemonics—Primarily intended for nonal-
 2004 phabetic scripts like Japanese and Chinese. These mnemonics will have
 2005 a unique lead-in and lead-out symbol.

2006 All mnemonics are given a long descriptive name, written in the reference coded
 2007 character set and taken from ISO 10646 {B11}, if possible.

2008 **F.4.4 Two-Character Mnemonics**

2009 The two-character mnemonics include various accented Latin letters, Greek,
 2010 Cyrillic, Hebrew, Arabic, Hiragana, Katakana, and Bopomofo. Some special char-
 2011 acters also are included. Almost all ISO or ISO-registered 7- and 8-bit coded char-
 2012 acter sets are covered with these two-character mnemonics.

2013 The two characters are chosen so the graphical appearance in the reference set
 2014 resembles as much as possible (within the possibilities available) the graphical
 2015 appearance of the character. The basic coded character set of ISO 646 {1} is used
 2016 as the reference set, as described previously.

2017 The characters in the reference coded character set are chosen to represent them-
 2018 selves. They may be considered as two-character mnemonics where the second
 2019 character is a space.

2020 Control character mnemonics are chosen according to ISO 2047 {B3} and ISO 6429
 2021 {B5}.

2022 Letters, including Greek, Cyrillic, Arabic, and Hebrew, are represented with the
 2023 base letter as the first letter, and the second letter represents an accent or rela-
 2024 tion to a non-Latin script. Non-Latin letters are transliterated to Latin letters,
 2025 following transliteration standards as closely as possible.

2026 After a letter, the second character signifies the following:

2027	exclamation-mark	!	grave
2028	apostrophe	'	acute accent
2029	greater-than-sign	>	circumflex accent

2030	question-mark	?	tilde
2031	hyphen-minus	–	macron
2032	left-parenthesis	(breve
2033	full-stop	.	dot above/ring above
2034	colon	:	diaeresis
2035	comma	,	cedilla
2036	underline	—	underline
2037	solidus	/	stroke
2038	quotation-mark	"	double acute accent
2039	semicolon	;	ogonek
2040	less-than-sign	<	caron
2041	equals	=	Cyrillian
2042	asterisk	*	Greek
2043	percent-sign	%	Greek/Cyrillian special
2044	plus	+	smalls: Arabic, capitals: Hebrew
2045	four	4	Bopomofo
2046	five	5	Hiragana
2047	six	6	Katakana

2048 Special characters are encoded with some mnemonic value. These are not sys-
 2049 tematic throughout, but most mnemonics start with a special character of the
 2050 reference set. Special characters with some sort of reference to the reference
 2051 coded character set normally have this character as the first character in the
 2052 mnemonic.

2053 **F.4.5 Variable-Length Character Mnemonics**

2054 The variable-length character mnemonics are meant primarily for the ideographic
 2055 characters in larger Asian coded character sets. To have the mnemonics as short
 2056 as possible, which both saves storage and is easier to type, a short name is pre-
 2057 ferred. Considering the Chinese standard GB 2312 {B14} and the Japanese stan-
 2058 dards JIS X0208 {B15} and JIS X0212 {B16}, they are all given by row and column
 2059 numbers between 1 and 99. So two positions for row and column and a coded
 2060 character set identifier of one character would be almost as short as possible. The
 2061 following coded character set identifiers are defined:

2062	c	GB 2312 {B14}
2063	j	JIS X0208 {B15}
2064	J	JIS X0212 {B16}
2065	k	KS C 5601 {B17}

2066 The first idea was to have a name in Latin describing the pronunciation, but that
 2067 is not possible according to Asian sources.

2068 The variable-length character mnemonics can also be used for some Latin letters
 2069 with more than one accent or other special characters that are used less fre-
 2070 quently.

F.5 (Example) Danish Charmap Files

The (example) Danish locale is coded character-set independent, as it is defined in terms of symbolic character names. Symbolic character names are defined for about 1 300 characters, covering many coded character sets. It is not necessary to have all these characters present in the actual encoding character set because absent characters simply can be ignored. But specifying the locale with symbolic character names ensures a uniform collating sequence of the present characters, regardless of the encoded character set. The more complicated locale should not imply less efficient code at running time, although generating the locale tables could take a longer time.

Danish Standards provides several charmap files, of which the ISO_10646 is the prime charmap, as it defines all the character names. It is expected, however, that the ISO_8859-1 charmap would be of more current interest. The charmaps are quite general, and might be used for other countries' locales without change.

See the guidelines for character mnemonics in F.4 for guidance in reading these charmap files.

F.5.1 iso_10646 Charmap

```
# ISO/IEC DIS 10646: 1990 charmap based on ISO/IEC JTC1/SC2/WG2 N666
# Only a part of the 10646 encoding is tabled here
```

```
<escape_char> /
<mb_cur_max> 4
CHARMAP
<NUL> /d000/d128/d128/d128 NULL (NUL) 1
<SOH> /d001/d128/d128/d128 START OF HEADING (SOH) 1
<STX> /d002/d128/d128/d128 START OF TEXT (STX) 1
<ETX> /d003/d128/d128/d128 END OF TEXT (ETX) 1
<EOT> /d004/d128/d128/d128 END OF TRANSMISSION (EOT) 1
<ENQ> /d005/d128/d128/d128 ENQUIRY (ENQ) 1
<ACK> /d006/d128/d128/d128 ACKNOWLEDGE (ACK) 1
<alert> /d007/d128/d128/d128 BELL (BEL) 1
<BEL> /d007/d128/d128/d128 BELL (BEL) 1
<backspace> /d008/d128/d128/d128 BACKSPACE (BS) 1
<tab> /d009/d128/d128/d128 CHARACTER TABULATION (HT) 1
<newline> /d010/d128/d128/d128 LINE FEED (LF) 1
<vertical-tab> /d011/d128/d128/d128 LINE TABULATION (VT) 1
<form-feed> /d012/d128/d128/d128 FORM FEED (FF) 1
<carriage-return> /d013/d128/d128/d128 CARRIAGE RETURN (CR) 1
<DLE> /d016/d128/d128/d128 DATALINK ESCAPE (DLE) 1
<DC1> /d017/d128/d128/d128 DEVICE CONTROL ONE (DC1) 1
<DC2> /d018/d128/d128/d128 DEVICE CONTROL TWO (DC2) 1
<DC3> /d019/d128/d128/d128 DEVICE CONTROL THREE (DC3) 1
<DC4> /d020/d128/d128/d128 DEVICE CONTROL FOUR (DC4) 1
<NAK> /d021/d128/d128/d128 NEGATIVE ACKNOWLEDGE (NAK) 1
<SYN> /d022/d128/d128/d128 SYNCHRONOUS IDLE (SYN) 1
<ETB> /d023/d128/d128/d128 END OF TRANSMISSION BLOCK (ETB) 1
<CAN> /d024/d128/d128/d128 CANCEL (CAN) 1
<SUB> /d026/d128/d128/d128 SUBSTITUTE (SUB) 1
<ESC> /d027/d128/d128/d128 ESCAPE (ESC) 1
```

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2119	<IS4>	/d028/d128/d128/d128	FILE SEPARATOR (IS4)	1
2120	<IS3>	/d029/d128/d128/d128	GROUP SEPARATOR (IS3)	1
2121	<intro>	/d029/d128/d128/d128	GROUP SEPARATOR (IS3)	1
2122	<IS2>	/d030/d128/d128/d128	RECORD SEPARATOR (IS2)	1
2123	<IS1>	/d031/d128/d128/d128	UNIT SEPARATOR (IS1)	1
2124		/d127/d128/d128/d128	DELETE (DEL)	1
2125	<space>	/d032/d032/d032/d032	SPACE	
2126	<exclamation-mark>	/d032/d032/d032/d033	EXCLAMATION MARK	
2127	<quotation-mark>	/d032/d032/d032/d034	QUOTATION MARK	
2128	<number-sign>	/d032/d032/d032/d035	NUMBER SIGN	
2129	<dollar-sign>	/d032/d032/d032/d036	DOLLAR SIGN	
2130	<percent-sign>	/d032/d032/d032/d037	PERCENT SIGN	
2131	<ampersand>	/d032/d032/d032/d038	AMPERSAND	
2132	<apostrophe>	/d032/d032/d032/d039	APOSTROPHE	
2133	<left-parenthesis>	/d032/d032/d032/d040	LEFT PARENTHESIS	
2134	<right-parenthesis>	/d032/d032/d032/d041	RIGHT PARENTHESIS	
2135	<asterisk>	/d032/d032/d032/d042	ASTERISK	
2136	<plus-sign>	/d032/d032/d032/d043	PLUS SIGN	
2137	<comma>	/d032/d032/d032/d044	COMMA	
2138	<hyphen>	/d032/d032/d032/d045	HYPHEN-MINUS	
2139	<hyphen-minus>	/d032/d032/d032/d045	HYPHEN-MINUS	
2140	<period>	/d032/d032/d032/d046	FULL STOP	
2141	<full-stop>	/d032/d032/d032/d046	FULL STOP	
2142	<slash>	/d032/d032/d032/d047	SOLIDUS	
2143	<solidus>	/d032/d032/d032/d047	SOLIDUS	
2144	<zero>	/d032/d032/d032/d048	DIGIT ZERO	
2145	<one>	/d032/d032/d032/d049	DIGIT ONE	
2146	<two>	/d032/d032/d032/d050	DIGIT TWO	
2147	<three>	/d032/d032/d032/d051	DIGIT THREE	
2148	<four>	/d032/d032/d032/d052	DIGIT FOUR	
2149	<five>	/d032/d032/d032/d053	DIGIT FIVE	
2150	<six>	/d032/d032/d032/d054	DIGIT SIX	
2151	<seven>	/d032/d032/d032/d055	DIGIT SEVEN	
2152	<eight>	/d032/d032/d032/d056	DIGIT EIGHT	
2153	<nine>	/d032/d032/d032/d057	DIGIT NINE	
2154	<colon>	/d032/d032/d032/d058	COLON	
2155	<semicolon>	/d032/d032/d032/d059	SEMICOLON	
2156	<less-than-sign>	/d032/d032/d032/d060	LESS-THAN SIGN	
2157	<equals-sign>	/d032/d032/d032/d061	EQUALS SIGN	
2158	<greater-than-sign>	/d032/d032/d032/d062	GREATER-THAN SIGN	
2159	<question-mark>	/d032/d032/d032/d063	QUESTION MARK	
2160	<commercial-at>	/d032/d032/d032/d064	COMMERCIAL AT	
2161	<left-square-bracket>	/d032/d032/d032/d091	LEFT SQUARE BRACKET	
2162	<reverse-solidus>	/d032/d032/d032/d092	REVERSE SOLIDUS	
2163	<backslash>	/d032/d032/d032/d092	REVERSE SOLIDUS	
2164	<right-square-bracket>	/d032/d032/d032/d093	RIGHT SQUARE BRACKET	
2165	<circumflex-accent>	/d032/d032/d032/d094	CIRCUMFLEX ACCENT	
2166	<low-line>	/d032/d032/d032/d095	LOW LINE	
2167	<underscore>	/d032/d032/d032/d095	LOW LINE	
2168	<grave-accent>	/d032/d032/d032/d096	GRAVE ACCENT	
2169	<left-curly-bracket>	/d032/d032/d032/d123	LEFT CURLY BRACKET	
2170	<vertical-line>	/d032/d032/d032/d124	VERTICAL LINE	
2171	<right-curly-bracket>	/d032/d032/d032/d125	RIGHT CURLY BRACKET	
2172	<tilde>	/d032/d032/d032/d126	TILDE	
2173	<SP>	/d032/d032/d032/d032	SPACE	
2174	<!>	/d032/d032/d032/d033	EXCLAMATION MARK	
2175	<">	/d032/d032/d032/d034	QUOTATION MARK	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2176	<Nb>	/d032/d032/d032/d035	NUMBER SIGN
2177	<DO>	/d032/d032/d032/d036	DOLLAR SIGN
2178	<%>	/d032/d032/d032/d037	PERCENT SIGN
2179	<&>	/d032/d032/d032/d038	AMPERSAND
2180	<'>	/d032/d032/d032/d039	APOSTROPHE
2181	<(>	/d032/d032/d032/d040	LEFT PARENTHESIS
2182	<)>	/d032/d032/d032/d041	RIGHT PARENTHESIS
2183	<*>	/d032/d032/d032/d042	ASTERISK
2184	<+>	/d032/d032/d032/d043	PLUS SIGN
2185	<,>	/d032/d032/d032/d044	COMMA
2186	<->	/d032/d032/d032/d045	HYPHEN-MINUS
2187	<.>	/d032/d032/d032/d046	FULL STOP
2188	</>	/d032/d032/d032/d047	SOLIDUS
2189	<0>	/d032/d032/d032/d048	DIGIT ZERO
2190	<1>	/d032/d032/d032/d049	DIGIT ONE
2191	<2>	/d032/d032/d032/d050	DIGIT TWO
2192	<3>	/d032/d032/d032/d051	DIGIT THREE
2193	<4>	/d032/d032/d032/d052	DIGIT FOUR
2194	<5>	/d032/d032/d032/d053	DIGIT FIVE
2195	<6>	/d032/d032/d032/d054	DIGIT SIX
2196	<7>	/d032/d032/d032/d055	DIGIT SEVEN
2197	<8>	/d032/d032/d032/d056	DIGIT EIGHT
2198	<9>	/d032/d032/d032/d057	DIGIT NINE
2199	<:>	/d032/d032/d032/d058	COLON
2200	<:>	/d032/d032/d032/d059	SEMICOLON
2201	<<>	/d032/d032/d032/d060	LESS-THAN SIGN
2202	<=>	/d032/d032/d032/d061	EQUALS SIGN
2203	</>>	/d032/d032/d032/d062	GREATER-THAN SIGN
2204	<?>	/d032/d032/d032/d063	QUESTION MARK
2205	<At>	/d032/d032/d032/d064	COMMERCIAL AT
2206	<A>	/d032/d032/d032/d065	LATIN CAPITAL LETTER A
2207		/d032/d032/d032/d066	LATIN CAPITAL LETTER B
2208	<C>	/d032/d032/d032/d067	LATIN CAPITAL LETTER C
2209	<D>	/d032/d032/d032/d068	LATIN CAPITAL LETTER D
2210	<E>	/d032/d032/d032/d069	LATIN CAPITAL LETTER E
2211	<F>	/d032/d032/d032/d070	LATIN CAPITAL LETTER F
2212	<G>	/d032/d032/d032/d071	LATIN CAPITAL LETTER G
2213	<H>	/d032/d032/d032/d072	LATIN CAPITAL LETTER H
2214	<I>	/d032/d032/d032/d073	LATIN CAPITAL LETTER I
2215	<J>	/d032/d032/d032/d074	LATIN CAPITAL LETTER J
2216	<K>	/d032/d032/d032/d075	LATIN CAPITAL LETTER K
2217	<L>	/d032/d032/d032/d076	LATIN CAPITAL LETTER L
2218	<M>	/d032/d032/d032/d077	LATIN CAPITAL LETTER M
2219	<N>	/d032/d032/d032/d078	LATIN CAPITAL LETTER N
2220	<O>	/d032/d032/d032/d079	LATIN CAPITAL LETTER O
2221	<P>	/d032/d032/d032/d080	LATIN CAPITAL LETTER P
2222	<Q>	/d032/d032/d032/d081	LATIN CAPITAL LETTER Q
2223	<R>	/d032/d032/d032/d082	LATIN CAPITAL LETTER R
2224	<S>	/d032/d032/d032/d083	LATIN CAPITAL LETTER S
2225	<T>	/d032/d032/d032/d084	LATIN CAPITAL LETTER T
2226	<U>	/d032/d032/d032/d085	LATIN CAPITAL LETTER U
2227	<V>	/d032/d032/d032/d086	LATIN CAPITAL LETTER V
2228	<W>	/d032/d032/d032/d087	LATIN CAPITAL LETTER W
2229	<X>	/d032/d032/d032/d088	LATIN CAPITAL LETTER X
2230	<Y>	/d032/d032/d032/d089	LATIN CAPITAL LETTER Y
2231	<Z>	/d032/d032/d032/d090	LATIN CAPITAL LETTER Z
2232	<<(>	/d032/d032/d032/d091	LEFT SQUARE BRACKET

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2233	<////>	/d032/d032/d032/d092	REVERSE SOLIDUS
2234	<)/>>	/d032/d032/d032/d093	RIGHT SQUARE BRACKET
2235	<' />>	/d032/d032/d032/d094	CIRCUMFLEX ACCENT
2236	<_>	/d032/d032/d032/d095	LOW LINE
2237	<' !>	/d032/d032/d032/d096	GRAVE ACCENT
2238	<a>	/d032/d032/d032/d097	LATIN SMALL LETTER A
2239		/d032/d032/d032/d098	LATIN SMALL LETTER B
2240	<c>	/d032/d032/d032/d099	LATIN SMALL LETTER C
2241	<d>	/d032/d032/d032/d100	LATIN SMALL LETTER D
2242	<e>	/d032/d032/d032/d101	LATIN SMALL LETTER E
2243	<f>	/d032/d032/d032/d102	LATIN SMALL LETTER F
2244	<g>	/d032/d032/d032/d103	LATIN SMALL LETTER G
2245	<h>	/d032/d032/d032/d104	LATIN SMALL LETTER H
2246	<i>	/d032/d032/d032/d105	LATIN SMALL LETTER I
2247	<j>	/d032/d032/d032/d106	LATIN SMALL LETTER J
2248	<k>	/d032/d032/d032/d107	LATIN SMALL LETTER K
2249	<l>	/d032/d032/d032/d108	LATIN SMALL LETTER L
2250	<m>	/d032/d032/d032/d109	LATIN SMALL LETTER M
2251	<n>	/d032/d032/d032/d110	LATIN SMALL LETTER N
2252	<o>	/d032/d032/d032/d111	LATIN SMALL LETTER O
2253	<p>	/d032/d032/d032/d112	LATIN SMALL LETTER P
2254	<q>	/d032/d032/d032/d113	LATIN SMALL LETTER Q
2255	<r>	/d032/d032/d032/d114	LATIN SMALL LETTER R
2256	<s>	/d032/d032/d032/d115	LATIN SMALL LETTER S
2257	<t>	/d032/d032/d032/d116	LATIN SMALL LETTER T
2258	<u>	/d032/d032/d032/d117	LATIN SMALL LETTER U
2259	<v>	/d032/d032/d032/d118	LATIN SMALL LETTER V
2260	<w>	/d032/d032/d032/d119	LATIN SMALL LETTER W
2261	<x>	/d032/d032/d032/d120	LATIN SMALL LETTER X
2262	<y>	/d032/d032/d032/d121	LATIN SMALL LETTER Y
2263	<z>	/d032/d032/d032/d122	LATIN SMALL LETTER Z
2264	<(!>	/d032/d032/d032/d123	LEFT CURLY BRACKET
2265	<! !>	/d032/d032/d032/d124	VERTICAL LINE
2266	<!)>	/d032/d032/d032/d125	RIGHT CURLY BRACKET
2267	<' ?>	/d032/d032/d032/d126	TILDE
2268	<NS>	/d032/d032/d032/d160	NO-BREAK SPACE
2269	<! I>	/d032/d032/d032/d161	INVERTED EXCLAMATION MARK
2270	<Ct>	/d032/d032/d032/d162	CENT SIGN
2271	<Pd>	/d032/d032/d032/d163	POUND SIGN
2272	<Cu>	/d032/d032/d032/d164	CURRENCY SIGN
2273	<Ye>	/d032/d032/d032/d165	YEN SIGN
2274	<BB>	/d032/d032/d032/d166	BROKEN BAR
2275	<SE>	/d032/d032/d032/d167	SECTION SIGN
2276	<' :>	/d032/d032/d032/d168	DIAERESIS
2277	<Co>	/d032/d032/d032/d169	COPYRIGHT SIGN
2278	<-a>	/d032/d032/d032/d170	FEMININE ORDINAL INDICATOR
2279	<<<>	/d032/d032/d032/d171	LEFT POINTING DOUBLE ANGLE QUOTATION MARK
2280	<NO>	/d032/d032/d032/d172	NOT SIGN
2281	<- ->	/d032/d032/d032/d173	SOFT HYPHEN
2282	<Rg>	/d032/d032/d032/d174	REGISTERED SIGN
2283	<' ->	/d032/d032/d032/d175	MACRON
2284	<DG>	/d032/d032/d032/d176	DEGREE SIGN
2285	<+ ->	/d032/d032/d032/d177	PLUS-MINUS SIGN
2286	<2S>	/d032/d032/d032/d178	SUPERSCRIFT TWO
2287	<3S>	/d032/d032/d032/d179	SUPERSCRIFT THREE
2288	<' ' >	/d032/d032/d032/d180	ACUTE ACCENT
2289	<My>	/d032/d032/d032/d181	MICRO SIGN

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2290	<PI>	/d032/d032/d032/d182	PILCROW SIGN	
2291	<.M>	/d032/d032/d032/d183	MIDDLE DOT	
2292	<'>	/d032/d032/d032/d184	CEDILLA	
2293	<1S>	/d032/d032/d032/d185	SUPERSCRIPIT ONE	
2294	<-o>	/d032/d032/d032/d186	MASCULINE ORDINAL INDICATOR	
2295	</>>>	/d032/d032/d032/d187	RIGHT POINTING DOUBLE ANGLE QUOTATION MARK	1
2296	<14>	/d032/d032/d032/d188	VULGAR FRACTION ONE QUARTER	
2297	<12>	/d032/d032/d032/d189	VULGAR FRACTION ONE HALF	
2298	<34>	/d032/d032/d032/d190	VULGAR FRACTION THREE QUARTERS	
2299	<?I>	/d032/d032/d032/d191	INVERTED QUESTION MARK	
2300	<A!>	/d032/d032/d032/d192	LATIN CAPITAL LETTER A WITH GRAVE	
2301	<A'>	/d032/d032/d032/d193	LATIN CAPITAL LETTER A WITH ACUTE	
2302	<A/>>	/d032/d032/d032/d194	LATIN CAPITAL LETTER A WITH CIRCUMFLEX	
2303	<A?>	/d032/d032/d032/d195	LATIN CAPITAL LETTER A WITH TILDE	
2304	<A:>	/d032/d032/d032/d196	LATIN CAPITAL LETTER A WITH DIAERESIS	
2305	<AA>	/d032/d032/d032/d197	LATIN CAPITAL LETTER A WITH RING ABOVE	
2306	<AE>	/d032/d032/d032/d198	LATIN CAPITAL LETTER AE	
2307	<C,>	/d032/d032/d032/d199	LATIN CAPITAL LETTER C WITH CEDILLA	
2308	<E!>	/d032/d032/d032/d200	LATIN CAPITAL LETTER E WITH GRAVE	
2309	<E'>	/d032/d032/d032/d201	LATIN CAPITAL LETTER E WITH ACUTE	
2310	<E/>>	/d032/d032/d032/d202	LATIN CAPITAL LETTER E WITH CIRCUMFLEX	
2311	<E:>	/d032/d032/d032/d203	LATIN CAPITAL LETTER E WITH DIAERESIS	
2312	<I!>	/d032/d032/d032/d204	LATIN CAPITAL LETTER I WITH GRAVE	
2313	<I'>	/d032/d032/d032/d205	LATIN CAPITAL LETTER I WITH ACUTE	
2314	<I/>>	/d032/d032/d032/d206	LATIN CAPITAL LETTER I WITH CIRCUMFLEX	
2315	<I:>	/d032/d032/d032/d207	LATIN CAPITAL LETTER I WITH DIAERESIS	
2316	<D->	/d032/d032/d032/d208	LATIN CAPITAL LETTER ETH (Icelandic)	
2317	<N?>	/d032/d032/d032/d209	LATIN CAPITAL LETTER N WITH TILDE	
2318	<O!>	/d032/d032/d032/d210	LATIN CAPITAL LETTER O WITH GRAVE	
2319	<O'>	/d032/d032/d032/d211	LATIN CAPITAL LETTER O WITH ACUTE	
2320	<O/>>	/d032/d032/d032/d212	LATIN CAPITAL LETTER O WITH CIRCUMFLEX	
2321	<O?>	/d032/d032/d032/d213	LATIN CAPITAL LETTER O WITH TILDE	
2322	<O:>	/d032/d032/d032/d214	LATIN CAPITAL LETTER O WITH DIAERESIS	
2323	<*X>	/d032/d032/d032/d215	MULTIPLICATION SIGN	
2324	<O//>	/d032/d032/d032/d216	LATIN CAPITAL LETTER O WITH STROKE	
2325	<U!>	/d032/d032/d032/d217	LATIN CAPITAL LETTER U WITH GRAVE	
2326	<U'>	/d032/d032/d032/d218	LATIN CAPITAL LETTER U WITH ACUTE	
2327	<U/>>	/d032/d032/d032/d219	LATIN CAPITAL LETTER U WITH CIRCUMFLEX	
2328	<U:>	/d032/d032/d032/d220	LATIN CAPITAL LETTER U WITH DIAERESIS	
2329	<Y'>	/d032/d032/d032/d221	LATIN CAPITAL LETTER Y WITH ACUTE	
2330	<TH>	/d032/d032/d032/d222	LATIN CAPITAL LETTER THORN (Icelandic)	
2331	<ss>	/d032/d032/d032/d223	LATIN SMALL LETTER SHARP S (German)	
2332	<a!>	/d032/d032/d032/d224	LATIN SMALL LETTER A WITH GRAVE	
2333	<a'>	/d032/d032/d032/d225	LATIN SMALL LETTER A WITH ACUTE	
2334	<a/>>	/d032/d032/d032/d226	LATIN SMALL LETTER A WITH CIRCUMFLEX	
2335	<a?>	/d032/d032/d032/d227	LATIN SMALL LETTER A WITH TILDE	
2336	<a:>	/d032/d032/d032/d228	LATIN SMALL LETTER A WITH DIAERESIS	
2337	<aa>	/d032/d032/d032/d229	LATIN SMALL LETTER A WITH RING ABOVE	
2338	<ae>	/d032/d032/d032/d230	LATIN SMALL LETTER AE	
2339	<c,>	/d032/d032/d032/d231	LATIN SMALL LETTER C WITH CEDILLA	
2340	<e!>	/d032/d032/d032/d232	LATIN SMALL LETTER E WITH GRAVE	
2341	<e'>	/d032/d032/d032/d233	LATIN SMALL LETTER E WITH ACUTE	
2342	<e/>>	/d032/d032/d032/d234	LATIN SMALL LETTER E WITH CIRCUMFLEX	
2343	<e:>	/d032/d032/d032/d235	LATIN SMALL LETTER E WITH DIAERESIS	
2344	<i!>	/d032/d032/d032/d236	LATIN SMALL LETTER I WITH GRAVE	
2345	<i'>	/d032/d032/d032/d237	LATIN SMALL LETTER I WITH ACUTE	
2346	<i/>>	/d032/d032/d032/d238	LATIN SMALL LETTER I WITH CIRCUMFLEX	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2347	<i:>	/d032/d032/d032/d239	LATIN SMALL LETTER I WITH DIAERESIS
2348	<d->	/d032/d032/d032/d240	LATIN SMALL LETTER ETH (Icelandic)
2349	<n?>	/d032/d032/d032/d241	LATIN SMALL LETTER N WITH TILDE
2350	<o!>	/d032/d032/d032/d242	LATIN SMALL LETTER O WITH GRAVE
2351	<o'>	/d032/d032/d032/d243	LATIN SMALL LETTER O WITH ACUTE
2352	<o/>>	/d032/d032/d032/d244	LATIN SMALL LETTER O WITH CIRCUMFLEX
2353	<o?>	/d032/d032/d032/d245	LATIN SMALL LETTER O WITH TILDE
2354	<o:>	/d032/d032/d032/d246	LATIN SMALL LETTER O WITH DIAERESIS
2355	<-:>	/d032/d032/d032/d247	DIVISION SIGN
2356	<o//>	/d032/d032/d032/d248	LATIN SMALL LETTER O WITH STROKE
2357	<u!>	/d032/d032/d032/d249	LATIN SMALL LETTER U WITH GRAVE
2358	<u'>	/d032/d032/d032/d250	LATIN SMALL LETTER U WITH ACUTE
2359	<u/>>	/d032/d032/d032/d251	LATIN SMALL LETTER U WITH CIRCUMFLEX
2360	<u:>	/d032/d032/d032/d252	LATIN SMALL LETTER U WITH DIAERESIS
2361	<y'>	/d032/d032/d032/d253	LATIN SMALL LETTER Y WITH ACUTE
2362	<th>	/d032/d032/d032/d254	LATIN SMALL LETTER THORN (Icelandic)
2363	<y:>	/d032/d032/d032/d255	LATIN SMALL LETTER Y WITH DIAERESIS
2364	<A->	/d032/d032/d033/d033	LATIN CAPITAL LETTER A WITH MACRON
2365	<C/>>	/d032/d032/d033/d034	LATIN CAPITAL LETTER C WITH CIRCUMFLEX
2366	<C.>	/d032/d032/d033/d035	LATIN CAPITAL LETTER C WITH DOT ABOVE
2367	<E->	/d032/d032/d033/d036	LATIN CAPITAL LETTER E WITH MACRON
2368	<E.>	/d032/d032/d033/d037	LATIN CAPITAL LETTER E WITH DOT ABOVE
2369	<G/>>	/d032/d032/d033/d039	LATIN CAPITAL LETTER G WITH CIRCUMFLEX
2370	<'6>	/d032/d032/d033/d041	LEFT SINGLE QUOTATION MARK
2371	<"6>	/d032/d032/d033/d042	LEFT DOUBLE QUOTATION MARK
2372	<G(>	/d032/d032/d033/d043	LATIN CAPITAL LETTER G WITH BREVE
2373	<<->	/d032/d032/d033/d044	LEFTWARD ARROW
2374	<-!>	/d032/d032/d033/d045	UPWARD ARROW
2375	<-/>>	/d032/d032/d033/d046	RIGHTWARD ARROW
2376	<-v>	/d032/d032/d033/d047	DOWNWARD ARROW
2377	<a->	/d032/d032/d033/d049	LATIN SMALL LETTER A WITH MACRON
2378	<c/>>	/d032/d032/d033/d050	LATIN SMALL LETTER C WITH CIRCUMFLEX
2379	<c.>	/d032/d032/d033/d051	LATIN SMALL LETTER C WITH DOT ABOVE
2380	<e->	/d032/d032/d033/d052	LATIN SMALL LETTER E WITH MACRON
2381	<e.>	/d032/d032/d033/d053	LATIN SMALL LETTER E WITH DOT ABOVE
2382	<g/>>	/d032/d032/d033/d055	LATIN SMALL LETTER G WITH CIRCUMFLEX
2383	<'9>	/d032/d032/d033/d057	RIGHT SINGLE QUOTATION MARK
2384	<"9>	/d032/d032/d033/d058	RIGHT DOUBLE QUOTATION MARK
2385	<g(>	/d032/d032/d033/d059	LATIN SMALL LETTER G WITH BREVE
2386	<G.>	/d032/d032/d033/d065	LATIN CAPITAL LETTER G WITH DOT ABOVE
2387	<G,>	/d032/d032/d033/d066	LATIN CAPITAL LETTER G WITH CEDILLA
2388	<H/>>	/d032/d032/d033/d067	LATIN CAPITAL LETTER H WITH CIRCUMFLEX
2389	<I?>	/d032/d032/d033/d070	LATIN CAPITAL LETTER I WITH TILDE
2390	<I->	/d032/d032/d033/d071	LATIN CAPITAL LETTER I WITH MACRON
2391	<I.>	/d032/d032/d033/d072	LATIN CAPITAL LETTER I WITH DOT ABOVE
2392	<'0>	/d032/d032/d033/d074	RING ABOVE
2393	<HB>	/d032/d032/d033/d080	HORIZONTAL BAR
2394	<g.>	/d032/d032/d033/d081	LATIN SMALL LETTER G WITH DOT ABOVE
2395	<g,>	/d032/d032/d033/d082	LATIN SMALL LETTER G WITH CEDILLA
2396	<h/>>	/d032/d032/d033/d083	LATIN SMALL LETTER H WITH CIRCUMFLEX
2397	<TM>	/d032/d032/d033/d084	TRADE MARK SIGN
2398	<Md>	/d032/d032/d033/d085	MUSIC NOTE
2399	<i?>	/d032/d032/d033/d086	LATIN SMALL LETTER I WITH TILDE
2400	<i->	/d032/d032/d033/d087	LATIN SMALL LETTER I WITH MACRON
2401	<18>	/d032/d032/d033/d092	VULGAR FRACTION ONE EIGHTH
2402	<38>	/d032/d032/d033/d093	VULGAR FRACTION THREE EIGHTHS
2403	<58>	/d032/d032/d033/d094	VULGAR FRACTION FIVE EIGHTHS

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2404	<78>	/d032/d032/d033/d095	VULGAR FRACTION SEVEN EIGHTHS
2405	<Om>	/d032/d032/d033/d096	OHM SIGN
2406	<I; >	/d032/d032/d033/d097	LATIN CAPITAL LETTER I WITH OGONEK
2407	<J/>>	/d032/d032/d033/d098	LATIN CAPITAL LETTER J WITH CIRCUMFLEX
2408	<K, >	/d032/d032/d033/d099	LATIN CAPITAL LETTER K WITH CEDILLA
2409	<H//>	/d032/d032/d033/d100	LATIN CAPITAL LETTER H WITH STROKE
2410	<IJ>	/d032/d032/d033/d102	LATIN CAPITAL LIGATURE IJ
2411	<L. >	/d032/d032/d033/d103	LATIN CAPITAL LETTER L WITH MIDDLE DOT
2412	<L, >	/d032/d032/d033/d104	LATIN CAPITAL LETTER L WITH CEDILLA
2413	<N, >	/d032/d032/d033/d105	LATIN CAPITAL LETTER N WITH CEDILLA
2414	<OE>	/d032/d032/d033/d106	LATIN CAPITAL LIGATURE OE
2415	<O->	/d032/d032/d033/d107	LATIN CAPITAL LETTER O WITH MACRON
2416	<T//>	/d032/d032/d033/d109	LATIN CAPITAL LETTER T WITH STROKE
2417	<NG>	/d032/d032/d033/d110	LATIN CAPITAL LETTER ENG (Lappish)
2418	<'n>	/d032/d032/d033/d111	LATIN SMALL LETTER N PRECEDED BY APOSTROPHE
2419	<kk>	/d032/d032/d033/d112	LATIN SMALL LETTER KRA (Greenlandic)
2420	<i; >	/d032/d032/d033/d113	LATIN SMALL LETTER I WITH OGONEK
2421	<j/>>	/d032/d032/d033/d114	LATIN SMALL LETTER J WITH CIRCUMFLEX
2422	<k, >	/d032/d032/d033/d115	LATIN SMALL LETTER K WITH CEDILLA
2423	<h//>	/d032/d032/d033/d116	LATIN SMALL LETTER H WITH STROKE
2424	<i. >	/d032/d032/d033/d117	LATIN SMALL LETTER I WITH NO DOT
2425	<ij>	/d032/d032/d033/d118	LATIN SMALL LIGATURE IJ
2426	<l. >	/d032/d032/d033/d119	LATIN SMALL LETTER L WITH MIDDLE DOT
2427	<l, >	/d032/d032/d033/d120	LATIN SMALL LETTER L WITH CEDILLA
2428	<n, >	/d032/d032/d033/d121	LATIN SMALL LETTER N WITH CEDILLA
2429	<oe>	/d032/d032/d033/d122	LATIN SMALL LIGATURE OE
2430	<o->	/d032/d032/d033/d123	LATIN SMALL LETTER O WITH MACRON
2431	<t//>	/d032/d032/d033/d125	LATIN SMALL LETTER T WITH STROKE
2432	<ng>	/d032/d032/d033/d126	LATIN SMALL LETTER ENG
2433	<A; >	/d032/d032/d033/d161	LATIN CAPITAL LETTER A WITH OGONEK
2434	<' (>	/d032/d032/d033/d162	BREVE
2435	<L//>	/d032/d032/d033/d163	LATIN CAPITAL LETTER L WITH STROKE
2436	<L<>	/d032/d032/d033/d165	LATIN CAPITAL LETTER L WITH CARON
2437	<S' >	/d032/d032/d033/d166	LATIN CAPITAL LETTER S WITH ACUTE
2438	<S/>>	/d032/d032/d033/d168	LATIN CAPITAL LETTER S WITH CIRCUMFLEX
2439	<S<>	/d032/d032/d033/d169	LATIN CAPITAL LETTER S WITH CARON
2440	<S, >	/d032/d032/d033/d170	LATIN CAPITAL LETTER S WITH CEDILLA
2441	<T<>	/d032/d032/d033/d171	LATIN CAPITAL LETTER T WITH CARON
2442	<Z' >	/d032/d032/d033/d172	LATIN CAPITAL LETTER Z WITH ACUTE
2443	<Z<>	/d032/d032/d033/d174	LATIN CAPITAL LETTER Z WITH CARON
2444	<Z. >	/d032/d032/d033/d175	LATIN CAPITAL LETTER Z WITH DOT ABOVE
2445	<a; >	/d032/d032/d033/d177	LATIN SMALL LETTER A WITH OGONEK
2446	<' ; >	/d032/d032/d033/d178	OGONEK
2447	<l//>	/d032/d032/d033/d179	LATIN SMALL LETTER L WITH STROKE
2448	<l<>	/d032/d032/d033/d181	LATIN SMALL LETTER L WITH CARON
2449	<s' >	/d032/d032/d033/d182	LATIN SMALL LETTER S WITH ACUTE
2450	<' <>	/d032/d032/d033/d183	CARON
2451	<s/>>	/d032/d032/d033/d184	LATIN SMALL LETTER S WITH CIRCUMFLEX
2452	<s<>	/d032/d032/d033/d185	LATIN SMALL LETTER S WITH CARON
2453	<s, >	/d032/d032/d033/d186	LATIN SMALL LETTER S WITH CEDILLA
2454	<t<>	/d032/d032/d033/d187	LATIN SMALL LETTER T WITH CARON
2455	<z' >	/d032/d032/d033/d188	LATIN SMALL LETTER Z WITH ACUTE
2456	<' ">	/d032/d032/d033/d189	DOUBLE ACUTE ACCENT
2457	<z<>	/d032/d032/d033/d190	LATIN SMALL LETTER Z WITH CARON
2458	<z. >	/d032/d032/d033/d191	LATIN SMALL LETTER Z WITH DOT ABOVE
2459	<R' >	/d032/d032/d033/d192	LATIN CAPITAL LETTER R WITH ACUTE
2460	<R, >	/d032/d032/d033/d193	LATIN CAPITAL LETTER R WITH CEDILLA

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2461	<A(>	/d032/d032/d033/d195	LATIN CAPITAL LETTER A WITH BREVE	
2462	<L'>	/d032/d032/d033/d197	LATIN CAPITAL LETTER L WITH ACUTE	
2463	<C'>	/d032/d032/d033/d198	LATIN CAPITAL LETTER C WITH ACUTE	
2464	<C<>	/d032/d032/d033/d200	LATIN CAPITAL LETTER C WITH CARON	
2465	<E;>	/d032/d032/d033/d202	LATIN CAPITAL LETTER E WITH OGONEK	
2466	<E<>	/d032/d032/d033/d204	LATIN CAPITAL LETTER E WITH CARON	
2467	<D<>	/d032/d032/d033/d207	LATIN CAPITAL LETTER D WITH CARON	
2468	<D//>	/d032/d032/d033/d208	LATIN CAPITAL LETTER D WITH STROKE	
2469	<N'>	/d032/d032/d033/d209	LATIN CAPITAL LETTER N WITH ACUTE	
2470	<N<>	/d032/d032/d033/d210	LATIN CAPITAL LETTER N WITH CARON	
2471	<U?>	/d032/d032/d033/d212	LATIN CAPITAL LETTER U WITH TILDE	
2472	<O">	/d032/d032/d033/d213	LATIN CAPITAL LETTER O WITH DOUBLE ACUTE	
2473	<U->	/d032/d032/d033/d214	LATIN CAPITAL LETTER U WITH MACRON	
2474	<U(>	/d032/d032/d033/d215	LATIN CAPITAL LETTER U WITH BREVE	
2475	<R<>	/d032/d032/d033/d216	LATIN CAPITAL LETTER R WITH CARON	
2476	<U0>	/d032/d032/d033/d217	LATIN CAPITAL LETTER U WITH RING ABOVE	1
2477	<U;>	/d032/d032/d033/d218	LATIN CAPITAL LETTER U WITH OGONEK	
2478	<U">	/d032/d032/d033/d219	LATIN CAPITAL LETTER U WITH DOUBLE ACUTE	
2479	<W/>>	/d032/d032/d033/d220	LATIN CAPITAL LETTER W WITH CIRCUMFLEX	
2480	<Y/>>	/d032/d032/d033/d221	LATIN CAPITAL LETTER Y WITH CIRCUMFLEX	
2481	<T,>	/d032/d032/d033/d222	LATIN CAPITAL LETTER T WITH CEDILLA	
2482	<Y:>	/d032/d032/d033/d223	LATIN CAPITAL LETTER Y WITH DIAERESIS	
2483	<r'>	/d032/d032/d033/d224	LATIN SMALL LETTER R WITH ACUTE	
2484	<r,>	/d032/d032/d033/d225	LATIN SMALL LETTER R WITH CEDILLA	
2485	<a(>	/d032/d032/d033/d227	LATIN SMALL LETTER A WITH BREVE	
2486	<l'>	/d032/d032/d033/d229	LATIN SMALL LETTER L WITH ACUTE	
2487	<c'>	/d032/d032/d033/d230	LATIN SMALL LETTER C WITH ACUTE	
2488	<c<>	/d032/d032/d033/d232	LATIN SMALL LETTER C WITH CARON	
2489	<e;>	/d032/d032/d033/d234	LATIN SMALL LETTER E WITH OGONEK	
2490	<e<>	/d032/d032/d033/d236	LATIN SMALL LETTER E WITH CARON	
2491	<d<>	/d032/d032/d033/d239	LATIN SMALL LETTER D WITH CARON	
2492	<d//>	/d032/d032/d033/d240	LATIN SMALL LETTER D WITH STROKE	
2493	<n'>	/d032/d032/d033/d241	LATIN SMALL LETTER N WITH ACUTE	
2494	<n<>	/d032/d032/d033/d242	LATIN SMALL LETTER N WITH CARON	
2495	<u?>	/d032/d032/d033/d244	LATIN SMALL LETTER U WITH TILDE	
2496	<o">	/d032/d032/d033/d245	LATIN SMALL LETTER O WITH DOUBLE ACUTE	
2497	<u->	/d032/d032/d033/d246	LATIN SMALL LETTER U WITH MACRON	
2498	<u(>	/d032/d032/d033/d247	LATIN SMALL LETTER U WITH BREVE	
2499	<r<>	/d032/d032/d033/d248	LATIN SMALL LETTER R WITH CARON	
2500	<u0>	/d032/d032/d033/d249	LATIN SMALL LETTER U WITH RING ABOVE	1
2501	<u;>	/d032/d032/d033/d250	LATIN SMALL LETTER U WITH OGONEK	
2502	<u">	/d032/d032/d033/d251	LATIN SMALL LETTER U WITH DOUBLE ACUTE	
2503	<w/>>	/d032/d032/d033/d252	LATIN SMALL LETTER W WITH CIRCUMFLEX	
2504	<y/>>	/d032/d032/d033/d253	LATIN SMALL LETTER Y WITH CIRCUMFLEX	
2505	<t,>	/d032/d032/d033/d254	LATIN SMALL LETTER T WITH CEDILLA	
2506	<'>	/d032/d032/d033/d255	DOT ABOVE	
2507	<a<>	/d032/d032/d034/d032	LATIN SMALL LETTER A WITH CARON	
2508	<A<>	/d032/d032/d034/d033	LATIN CAPITAL LETTER A WITH CARON	
2509	<a_>	/d032/d032/d034/d034	LATIN SMALL LETTER A WITH LINE BELOW	
2510	<A_>	/d032/d032/d034/d035	LATIN CAPITAL LETTER A WITH LINE BELOW	
2511	<'a>	/d032/d032/d034/d048	LATIN SMALL LETTER A PRECEDED BY APOSTROPHE	
2512	<'A>	/d032/d032/d034/d049	LATIN CAPITAL LETTER A PRECEDED BY APOSTROPHE	
2513	<a1>	/d032/d032/d034/d052	LATIN SMALL LETTER A WITH MACRON AND DIAERESIS	
2514	<A1>	/d032/d032/d034/d053	LATIN CAPITAL LETTER A WITH MACRON AND DIAERESIS	
2515	<a2>	/d032/d032/d034/d054	LATIN SMALL LETTER A WITH MACRON AND DOT ABOVE	
2516	<A2>	/d032/d032/d034/d055	LATIN CAPITAL LETTER A WITH MACRON AND DOT ABOVE	
2517	<a3>	/d032/d032/d034/d056	LATIN SMALL LETTER AE WITH MACRON	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2518	<A3>	/d032/d032/d034/d057	LATIN CAPITAL LETTER AE WITH MACRON
2519	<b.>	/d032/d032/d034/d086	LATIN SMALL LETTER B WITH DOT ABOVE
2520	<B.>	/d032/d032/d034/d087	LATIN CAPITAL LETTER B WITH DOT ABOVE
2521	<b_>	/d032/d032/d034/d088	LATIN SMALL LETTER B WITH LINE BELOW
2522	<B_>	/d032/d032/d034/d089	LATIN CAPITAL LETTER B WITH LINE BELOW
2523	<d_>	/d032/d032/d034/d096	LATIN SMALL LETTER D WITH LINE BELOW
2524	<D_>	/d032/d032/d034/d097	LATIN CAPITAL LETTER D WITH LINE BELOW
2525	<d.>	/d032/d032/d034/d098	LATIN SMALL LETTER D WITH DOT BELOW
2526	<D.>	/d032/d032/d034/d099	LATIN CAPITAL LETTER D WITH DOT BELOW
2527	<d; >	/d032/d032/d034/d100	LATIN SMALL LETTER D WITH OGONEK
2528	<D; >	/d032/d032/d034/d101	LATIN CAPITAL LETTER D WITH OGONEK
2529	<e(>	/d032/d032/d034/d106	LATIN SMALL LETTER E WITH BREVE
2530	<E(>	/d032/d032/d034/d107	LATIN CAPITAL LETTER E WITH BREVE
2531	<e_>	/d032/d032/d034/d108	LATIN SMALL LETTER E WITH LINE BELOW
2532	<E_>	/d032/d032/d034/d109	LATIN CAPITAL LETTER E WITH LINE BELOW
2533	<;S>	/d032/d032/d034/d126	HIGH OGONEK
2534	<e?>	/d032/d032/d034/d168	LATIN SMALL LETTER E WITH TILDE
2535	<E?>	/d032/d032/d034/d169	LATIN CAPITAL LETTER E WITH TILDE
2536	<f.>	/d032/d032/d034/d180	LATIN SMALL LETTER F WITH DOT ABOVE
2537	<F.>	/d032/d032/d034/d181	LATIN CAPITAL LETTER F WITH DOT ABOVE
2538	<g<>	/d032/d032/d034/d182	LATIN SMALL LETTER G WITH CARON
2539	<G<>	/d032/d032/d034/d183	LATIN CAPITAL LETTER G WITH CARON
2540	<g->	/d032/d032/d034/d184	LATIN SMALL LETTER G WITH MACRON
2541	<G->	/d032/d032/d034/d185	LATIN CAPITAL LETTER G WITH MACRON
2542	<g//>	/d032/d032/d034/d188	LATIN SMALL LETTER G WITH STROKE
2543	<G//>	/d032/d032/d034/d189	LATIN CAPITAL LETTER G WITH STROKE
2544	<h:>	/d032/d032/d034/d192	LATIN SMALL LETTER H WITH DIAERESIS
2545	<H:>	/d032/d032/d034/d193	LATIN CAPITAL LETTER H WITH DIAERESIS
2546	<h.>	/d032/d032/d034/d194	LATIN SMALL LETTER H WITH DOT ABOVE
2547	<H.>	/d032/d032/d034/d195	LATIN CAPITAL LETTER H WITH DOT ABOVE
2548	<h,>	/d032/d032/d034/d196	LATIN SMALL LETTER H WITH CEDILLA
2549	<H,>	/d032/d032/d034/d197	LATIN CAPITAL LETTER H WITH CEDILLA
2550	<h; >	/d032/d032/d034/d198	LATIN SMALL LETTER H WITH OGONEK
2551	<H; >	/d032/d032/d034/d199	LATIN CAPITAL LETTER H WITH OGONEK
2552	<i<>	/d032/d032/d034/d204	LATIN SMALL LETTER I WITH CARON
2553	<I<>	/d032/d032/d034/d205	LATIN CAPITAL LETTER I WITH CARON
2554	<i(>	/d032/d032/d034/d206	LATIN SMALL LETTER I WITH BREVE
2555	<I(>	/d032/d032/d034/d207	LATIN CAPITAL LETTER I WITH BREVE
2556	<j(>	/d032/d032/d034/d224	LATIN SMALL LETTER J WITH BREVE
2557	<J(>	/d032/d032/d034/d225	LATIN CAPITAL LETTER J WITH BREVE
2558	<k'>	/d032/d032/d034/d226	LATIN SMALL LETTER K WITH ACUTE
2559	<K'>	/d032/d032/d034/d227	LATIN CAPITAL LETTER K WITH ACUTE
2560	<k<>	/d032/d032/d034/d228	LATIN SMALL LETTER K WITH CARON
2561	<K<>	/d032/d032/d034/d229	LATIN CAPITAL LETTER K WITH CARON
2562	<k_>	/d032/d032/d034/d230	LATIN SMALL LETTER K WITH LINE BELOW
2563	<K_>	/d032/d032/d034/d231	LATIN CAPITAL LETTER K WITH LINE BELOW
2564	<k.>	/d032/d032/d034/d232	LATIN SMALL LETTER K WITH DOT BELOW
2565	<K.>	/d032/d032/d034/d233	LATIN CAPITAL LETTER K WITH DOT BELOW
2566	<k; >	/d032/d032/d034/d234	LATIN SMALL LETTER K WITH OGONEK
2567	<K; >	/d032/d032/d034/d235	LATIN CAPITAL LETTER K WITH OGONEK
2568	<l_>	/d032/d032/d034/d240	LATIN SMALL LETTER L WITH LINE BELOW
2569	<L_>	/d032/d032/d034/d241	LATIN CAPITAL LETTER L WITH LINE BELOW
2570	<m'>	/d032/d032/d034/d248	LATIN SMALL LETTER M WITH ACUTE
2571	<M'>	/d032/d032/d034/d249	LATIN CAPITAL LETTER M WITH ACUTE
2572	<m.>	/d032/d032/d034/d250	LATIN SMALL LETTER M WITH DOT ABOVE
2573	<M.>	/d032/d032/d034/d251	LATIN CAPITAL LETTER M WITH DOT ABOVE
2574	<n.>	/d032/d032/d035/d034	LATIN SMALL LETTER N WITH DOT ABOVE

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2575	<N.>	/d032/d032/d035/d035	LATIN CAPITAL LETTER N WITH DOT ABOVE
2576	<n_>	/d032/d032/d035/d038	LATIN SMALL LETTER N WITH LINE BELOW
2577	<N_>	/d032/d032/d035/d039	LATIN CAPITAL LETTER N WITH LINE BELOW
2578	<o<>	/d032/d032/d035/d046	LATIN SMALL LETTER O WITH CARON
2579	<O<>	/d032/d032/d035/d047	LATIN CAPITAL LETTER O WITH CARON
2580	<o(>	/d032/d032/d035/d048	LATIN SMALL LETTER O WITH BREVE
2581	<O(>	/d032/d032/d035/d049	LATIN CAPITAL LETTER O WITH BREVE
2582	<o_>	/d032/d032/d035/d050	LATIN SMALL LETTER O WITH LINE BELOW
2583	<O_>	/d032/d032/d035/d051	LATIN CAPITAL LETTER O WITH LINE BELOW
2584	<o; >	/d032/d032/d035/d064	LATIN SMALL LETTER O WITH OGONEK
2585	<O; >	/d032/d032/d035/d065	LATIN CAPITAL LETTER O WITH OGONEK
2586	<o!>	/d032/d032/d035/d068	LATIN SMALL LETTER O WITH MACRON AND OGONEK
2587	<O!>	/d032/d032/d035/d069	LATIN CAPITAL LETTER O WITH MACRON AND OGONEK
2588	<p'>	/d032/d032/d035/d098	LATIN SMALL LETTER P WITH ACUTE
2589	<P'>	/d032/d032/d035/d099	LATIN CAPITAL LETTER P WITH ACUTE
2590	<r.>	/d032/d032/d035/d100	LATIN SMALL LETTER R WITH DOT ABOVE
2591	<R.>	/d032/d032/d035/d101	LATIN CAPITAL LETTER R WITH DOT ABOVE
2592	<r_>	/d032/d032/d035/d102	LATIN SMALL LETTER R WITH LINE BELOW
2593	<R_>	/d032/d032/d035/d103	LATIN CAPITAL LETTER R WITH LINE BELOW
2594	<s.>	/d032/d032/d035/d110	LATIN SMALL LETTER S WITH DOT ABOVE
2595	<S.>	/d032/d032/d035/d111	LATIN CAPITAL LETTER S WITH DOT ABOVE
2596	<s; >	/d032/d032/d035/d114	LATIN SMALL LETTER S WITH OGONEK
2597	<S; >	/d032/d032/d035/d115	LATIN CAPITAL LETTER S WITH OGONEK
2598	<t_>	/d032/d032/d035/d160	LATIN SMALL LETTER T WITH LINE BELOW
2599	<T_>	/d032/d032/d035/d161	LATIN CAPITAL LETTER T WITH LINE BELOW
2600	<t.>	/d032/d032/d035/d162	LATIN SMALL LETTER T WITH DOT BELOW
2601	<T.>	/d032/d032/d035/d163	LATIN CAPITAL LETTER T WITH DOT BELOW
2602	<u<>	/d032/d032/d035/d170	LATIN SMALL LETTER U WITH CARON
2603	<U<>	/d032/d032/d035/d171	LATIN CAPITAL LETTER U WITH CARON
2604	<v?>	/d032/d032/d035/d214	LATIN SMALL LETTER V WITH TILDE
2605	<V?>	/d032/d032/d035/d215	LATIN CAPITAL LETTER V WITH TILDE
2606	<w'>	/d032/d032/d035/d220	LATIN SMALL LETTER W WITH ACUTE
2607	<W'>	/d032/d032/d035/d221	LATIN CAPITAL LETTER W WITH ACUTE
2608	<w.>	/d032/d032/d035/d222	LATIN SMALL LETTER W WITH DOT ABOVE
2609	<W.>	/d032/d032/d035/d223	LATIN CAPITAL LETTER W WITH DOT ABOVE
2610	<w:>	/d032/d032/d035/d224	LATIN SMALL LETTER W WITH DIAERESIS
2611	<W:>	/d032/d032/d035/d225	LATIN CAPITAL LETTER W WITH DIAERESIS
2612	<x.>	/d032/d032/d035/d230	LATIN SMALL LETTER X WITH DOT ABOVE
2613	<X.>	/d032/d032/d035/d231	LATIN CAPITAL LETTER X WITH DOT ABOVE
2614	<x:>	/d032/d032/d035/d232	LATIN SMALL LETTER X WITH DIAERESIS
2615	<X:>	/d032/d032/d035/d233	LATIN CAPITAL LETTER X WITH DIAERESIS
2616	<y!>	/d032/d032/d035/d236	LATIN SMALL LETTER Y WITH GRAVE
2617	<Y!>	/d032/d032/d035/d237	LATIN CAPITAL LETTER Y WITH GRAVE
2618	<y.>	/d032/d032/d035/d238	LATIN SMALL LETTER Y WITH DOT ABOVE
2619	<Y.>	/d032/d032/d035/d239	LATIN CAPITAL LETTER Y WITH DOT ABOVE
2620	<z/>>	/d032/d032/d035/d244	LATIN SMALL LETTER Z WITH CIRCUMFLEX
2621	<Z/>>	/d032/d032/d035/d245	LATIN CAPITAL LETTER Z WITH CIRCUMFLEX
2622	<z(>	/d032/d032/d035/d246	LATIN SMALL LETTER Z WITH BREVE
2623	<Z(>	/d032/d032/d035/d247	LATIN CAPITAL LETTER Z WITH BREVE
2624	<z_>	/d032/d032/d035/d248	LATIN SMALL LETTER Z WITH LINE BELOW
2625	<Z_>	/d032/d032/d035/d249	LATIN CAPITAL LETTER Z WITH LINE BELOW
2626	<z//>	/d032/d032/d035/d252	LATIN SMALL LETTER Z WITH STROKE
2627	<Z//>	/d032/d032/d035/d253	LATIN CAPITAL LETTER Z WITH STROKE
2628	<ez>	/d032/d032/d035/d254	LATIN SMALL LETTER EZH WITH CARON
2629	<EZ>	/d032/d032/d035/d255	LATIN CAPITAL LETTER EZH WITH CARON
2630	<g'>	/d032/d032/d036/d033	LATIN SMALL LETTER G WITH ACUTE
2631	<G'>	/d032/d032/d036/d034	LATIN CAPITAL LETTER G WITH ACUTE

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2632	<'b>	/d032/d032/d036/d084	LATIN SMALL LETTER B PRECEDED BY APOSTROPHE
2633	<'B>	/d032/d032/d036/d085	LATIN CAPITAL LETTER B PRECEDED BY APOSTROPHE
2634	<'d>	/d032/d032/d036/d096	LATIN SMALL LETTER D PRECEDED BY APOSTROPHE
2635	<'D>	/d032/d032/d036/d097	LATIN CAPITAL LETTER D PRECEDED BY APOSTROPHE
2636	<'g>	/d032/d032/d036/d162	LATIN SMALL LETTER G PRECEDED BY APOSTROPHE
2637	<'G>	/d032/d032/d036/d163	LATIN CAPITAL LETTER G PRECEDED BY APOSTROPHE
2638	<'j>	/d032/d032/d036/d174	LATIN SMALL LETTER J PRECEDED BY APOSTROPHE
2639	<'J>	/d032/d032/d036/d175	LATIN CAPITAL LETTER J PRECEDED BY APOSTROPHE
2640	<'y>	/d032/d032/d036/d235	LATIN SMALL LETTER Y PRECEDED BY APOSTROPHE
2641	<'Y>	/d032/d032/d036/d236	LATIN CAPITAL LETTER Y PRECEDED BY APOSTROPHE
2642	<ed>	/d032/d032/d036/d239	LATIN SMALL LETTER EDZ
2643	<ED>	/d032/d032/d036/d240	LATIN CAPITAL LETTER EDZ
2644	<Vs>	/d032/d032/d037/d032	SPACE SYMBOL
2645	<1M>	/d032/d032/d037/d033	EM-SPACE
2646	<1N>	/d032/d032/d037/d034	EN-SPACE
2647	<3M>	/d032/d032/d037/d035	THREE-PER-EM SPACE
2648	<4M>	/d032/d032/d037/d036	FOUR-PER-EM SPACE
2649	<6M>	/d032/d032/d037/d037	SIX-PER-EM SPACE
2650	<1H>	/d032/d032/d037/d038	HAIR SPACE
2651	<1T>	/d032/d032/d037/d039	THIN SPACE
2652	<-1>	/d032/d032/d037/d040	HYPHEN
2653	<-N>	/d032/d032/d037/d041	EN-DASH
2654	<-2>	/d032/d032/d037/d042	MINUS SIGN
2655	<-M>	/d032/d032/d037/d043	EM-DASH
2656	<-3>	/d032/d032/d037/d044	QUOTATION DASH
2657	<'1>	/d032/d032/d037/d045	SINGLE PRIME
2658	<'2>	/d032/d032/d037/d046	DOUBLE PRIME
2659	<'3>	/d032/d032/d037/d047	TRIPLE PRIME
2660	<9'>	/d032/d032/d037/d048	SINGLE HIGH-REVERSED-9 QUOTATION MARK
2661	<9">	/d032/d032/d037/d049	DOUBLE HIGH-REVERSED-9 QUOTATION MARK
2662	<.9>	/d032/d032/d037/d050	SINGLE LOW-9 QUOTATION MARK
2663	<:9>	/d032/d032/d037/d051	DOUBLE LOW-9 QUOTATION MARK
2664	<<1>	/d032/d032/d037/d052	SINGLE LEFT-POINTING ANGLE QUOTATION MARK
2665	</>1>	/d032/d032/d037/d053	SINGLE RIGHT-POINTING ANGLE QUOTATION MARK
2666	<</>	/d032/d032/d037/d054	LEFT-POINTING ANGLE BRACKET
2667	<///>>	/d032/d032/d037/d055	RIGHT-POINTING ANGLE BRACKET
2668	<15>	/d032/d032/d037/d056	VULGAR FRACTION ONE FIFTH
2669	<25>	/d032/d032/d037/d057	VULGAR FRACTION TWO FIFTHS
2670	<35>	/d032/d032/d037/d058	VULGAR FRACTION THREE FIFTHS
2671	<45>	/d032/d032/d037/d059	VULGAR FRACTION FOUR FIFTHS
2672	<16>	/d032/d032/d037/d060	VULGAR FRACTION ONE SIXTH
2673	<13>	/d032/d032/d037/d061	VULGAR FRACTION ONE THIRD
2674	<23>	/d032/d032/d037/d062	VULGAR FRACTION TWO THIRDS
2675	<56>	/d032/d032/d037/d063	VULGAR FRACTION FIVE SIXTHS
2676	<*->	/d032/d032/d037/d064	MIDDLE ASTERISK
2677	<//->	/d032/d032/d037/d065	DAGGER
2678	<//=>	/d032/d032/d037/d066	DOUBLE-DAGGER
2679	<-X>	/d032/d032/d037/d067	MALTESE CROSS
2680	<%0>	/d032/d032/d037/d068	PER-MILLE SIGN
2681	<co>	/d032/d032/d037/d069	CARE-OF SIGN
2682	<PO>	/d032/d032/d037/d070	SOUND RECORDING COPYRIGHT SIGN
2683	<Rx>	/d032/d032/d037/d071	PRESCRIPTION SIGN
2684	<AO>	/d032/d032/d037/d072	ANGSTROM SIGN
2685	<oC>	/d032/d032/d037/d073	CENTIGRADE DEGREE SIGN
2686	<M1>	/d032/d032/d037/d074	MALE SIGN
2687	<Fm>	/d032/d032/d037/d075	FEMALE SIGN
2688	<T1>	/d032/d032/d037/d076	TELEPHONE SIGN

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2689	<TR>	/d032/d032/d037/d077	TELEPHONE RECORDER SIGN
2690	<MX>	/d032/d032/d037/d078	MUSICAL SHARP SIGN
2691	<Mb>	/d032/d032/d037/d079	MUSICAL FLAT SIGN
2692	<Mx>	/d032/d032/d037/d080	MUSICAL NATURAL SIGN
2693	<XX>	/d032/d032/d037/d081	BALLOT CROSS SIGN
2694	<OK>	/d032/d032/d037/d082	CHECK MARK
2695	<M2>	/d032/d032/d037/d083	DOUBLE MUSICAL NOTES
2696	<!2>	/d032/d032/d037/d084	DOUBLE EXCLAMATION MARKS
2697	<=2>	/d032/d032/d037/d085	DOUBLE LOW LINE
2698	<Ca>	/d032/d032/d037/d086	CARET
2699	<..>	/d032/d032/d037/d087	TWO-DOT LEADER
2700	<.3>	/d032/d032/d037/d088	HORIZONTAL ELLIPSIS
2701	<.3>	/d032/d032/d037/d089	VERTICAL ELLIPSIS
2702	<.:>	/d032/d032/d037/d090	THEREFORE SIGN
2703	<.:>	/d032/d032/d037/d091	BECAUSE SIGN
2704	<-+>	/d032/d032/d037/d092	MINUS-PLUS SIGN
2705	<!=>	/d032/d032/d037/d093	NOT EQUAL-TO SIGN
2706	<=3>	/d032/d032/d037/d094	IDENTICAL-TO SIGN
2707	<?1>	/d032/d032/d037/d095	DIFFERENCE-BETWEEN SIGN
2708	<?2>	/d032/d032/d037/d096	ALMOST-EQUALS SIGN
2709	<?->	/d032/d032/d037/d097	ASYMTOTICALLY-EQUALS SIGN
2710	<?=>	/d032/d032/d037/d098	SIMILAR-TO SIGN
2711	<=<>	/d032/d032/d037/d099	LESS-THAN OR EQUAL-TO SIGN
2712	</>=>	/d032/d032/d037/d100	GREATER-THAN OR EQUAL-TO SIGN
2713	<0(>	/d032/d032/d037/d101	PROPORTIONAL-TO SIGN
2714	<00>	/d032/d032/d037/d102	INFINITY SIGN
2715	<PP>	/d032/d032/d037/d103	PARALLEL-TO SIGN
2716	<-T>	/d032/d032/d037/d104	ORTHOGONAL-TO SIGN
2717	<-L>	/d032/d032/d037/d105	RIGHT ANGLE SIGN
2718	<-V>	/d032/d032/d037/d106	ANGLE SIGN
2719	<AN>	/d032/d032/d037/d107	LOGICAL-AND SIGN
2720	<OR>	/d032/d032/d037/d108	LOGICAL-OR SIGN
2721	<.P>	/d032/d032/d037/d109	PRODUCT DOT SIGN
2722	<nS>	/d032/d032/d037/d110	SUPERSCRITPT LATIN SMALL LETTER N
2723	<dP>	/d032/d032/d037/d111	PARTIAL DIFFERENTIAL SIGN
2724	<f(>	/d032/d032/d037/d112	FUNCTION SIGN
2725	<In>	/d032/d032/d037/d113	INTEGRAL SIGN
2726	<Io>	/d032/d032/d037/d114	CONTOUR INTEGRAL SIGN
2727	<RT>	/d032/d032/d037/d117	RADICAL SIGN
2728	<*P>	/d032/d032/d037/d118	REPEATED PRODUCT SIGN
2729	<+Z>	/d032/d032/d037/d119	SUMMATION SIGN
2730	<FA>	/d032/d032/d037/d120	FOR-ALL SIGN
2731	<TE>	/d032/d032/d037/d121	THERE-EXISTS SIGN
2732	<GF>	/d032/d032/d037/d122	GAMMA FUNCTION SIGN
2733	<DE>	/d032/d032/d037/d123	INCREMENT SIGN
2734	<NB>	/d032/d032/d037/d124	NABLA
2735	<(U>	/d032/d032/d037/d125	INTERSECTION SIGN
2736	<)U>	/d032/d032/d037/d126	UNION SIGN
2737	<(C>	/d032/d032/d037/d160	PROPER SUBSET SIGN
2738	<)C>	/d032/d032/d037/d161	PROPER SUPERSET SIGN
2739	<(_>	/d032/d032/d037/d162	SUBSET SIGN
2740	<)_>	/d032/d032/d037/d163	SUPERSET SIGN
2741	<(->	/d032/d032/d037/d164	ELEMENT-OF SIGN
2742	<-)>	/d032/d032/d037/d165	HAS AN ELEMENT SIGN
2743	<</>>	/d032/d032/d037/d166	LEFT AND RIGHT-POINTING ARROW
2744	<UD>	/d032/d032/d037/d167	UP AND DOWN-POINTING ARROW
2745	<Ub>	/d032/d032/d037/d168	UP AND DOWN-POINTING ARROW WITH LINE BELOW

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2746	<=>	/d032/d032/d037/d169	IMPLIED-BY SIGN
2747	<=/>	/d032/d032/d037/d170	IMPLIES SIGN
2748	<==>	/d032/d032/d037/d171	IF-AND-ONLY-IF SIGN
2749	</0>	/d032/d032/d037/d172	EMPTY SIGN
2750		/d032/d032/d037/d173	SOLID LOZENGE
2751	<0u>	/d032/d032/d037/d176	SMILING FACE WHITE
2752	<0U>	/d032/d032/d037/d177	SMILING FACE BLACK
2753	<SU>	/d032/d032/d037/d178	RADIANT SUN
2754	<0:>	/d032/d032/d037/d179	DOTTED CIRCLE
2755	<OS>	/d032/d032/d037/d180	SQUARE EMPTY
2756	<fS>	/d032/d032/d037/d181	SQUARE SOLID
2757	<Or>	/d032/d032/d037/d182	RECTANGLE EMPTY
2758	<SR>	/d032/d032/d037/d183	RECTANGLE SOLID
2759	<uT>	/d032/d032/d037/d184	UPWARDS-POINTING TRIANGLE EMPTY
2760	<UT>	/d032/d032/d037/d185	UPWARDS-POINTING TRIANGLE SOLID
2761	<dT>	/d032/d032/d037/d186	DOWNWARDS-POINTING TRIANGLE EMPTY
2762	<Dt>	/d032/d032/d037/d187	DOWNWARDS-POINTING TRIANGLE SOLID
2763	<PL>	/d032/d032/d037/d188	LEFTWARDS POINTER SOLID
2764	<PR>	/d032/d032/d037/d189	RIGHTWARDS POINTER SOLID
2765	<*1>	/d032/d032/d037/d190	STAR EMPTY
2766	<*2>	/d032/d032/d037/d191	STAR SOLID
2767	<VV>	/d032/d032/d037/d192	BOX DRAWINGS HEAVY VERTICAL
2768	<HH>	/d032/d032/d037/d193	BOX DRAWINGS HEAVY HORIZONTAL
2769	<DR>	/d032/d032/d037/d194	BOX DRAWINGS HEAVY DOWN AND RIGHT
2770	<LD>	/d032/d032/d037/d195	BOX DRAWINGS HEAVY DOWN AND LEFT
2771	<UR>	/d032/d032/d037/d196	BOX DRAWINGS HEAVY UP AND RIGHT
2772		/d032/d032/d037/d197	BOX DRAWINGS HEAVY UP AND LEFT
2773	<VR>	/d032/d032/d037/d198	BOX DRAWINGS HEAVY VERTICAL AND RIGHT
2774	<VL>	/d032/d032/d037/d199	BOX DRAWINGS HEAVY VERTICAL AND LEFT
2775	<DH>	/d032/d032/d037/d200	BOX DRAWINGS HEAVY HORIZONTAL AND DOWN
2776	<UH>	/d032/d032/d037/d201	BOX DRAWINGS HEAVY HORIZONTAL AND UP
2777	<VH>	/d032/d032/d037/d202	BOX DRAWINGS HEAVY VERTICAL AND HORIZONTAL
2778	<TB>	/d032/d032/d037/d203	BOX DRAWING SOLID UPPER HALF BLOCK
2779	<LB>	/d032/d032/d037/d204	BOX DRAWING SOLID LOWER HALF BLOCK
2780	<FB>	/d032/d032/d037/d205	BOX DRAWING SOLID FULL BLOCK
2781	<sB>	/d032/d032/d037/d206	BOX DRAWING SOLID SMALL SQUARE
2782	<EH>	/d032/d032/d037/d207	EMPTY HOUSE SIGN
2783	<vv>	/d032/d032/d037/d208	BOX DRAWINGS LIGHT VERTICAL
2784	<hh>	/d032/d032/d037/d209	BOX DRAWINGS LIGHT HORIZONTAL
2785	<dr>	/d032/d032/d037/d210	BOX DRAWINGS LIGHT DOWN AND RIGHT
2786	<dl>	/d032/d032/d037/d211	BOX DRAWINGS LIGHT DOWN AND LEFT
2787	<ur>	/d032/d032/d037/d212	BOX DRAWINGS LIGHT UP AND RIGHT
2788		/d032/d032/d037/d213	BOX DRAWINGS LIGHT UP AND LEFT
2789	<vr>	/d032/d032/d037/d214	BOX DRAWINGS LIGHT VERTICAL AND RIGHT
2790	<vl>	/d032/d032/d037/d215	BOX DRAWINGS LIGHT VERTICAL AND LEFT
2791	<dh>	/d032/d032/d037/d216	BOX DRAWINGS LIGHT HORIZONTAL AND DOWN
2792	<uh>	/d032/d032/d037/d217	BOX DRAWINGS LIGHT HORIZONTAL AND UP
2793	<vh>	/d032/d032/d037/d218	BOX DRAWINGS LIGHT VERTICAL AND HORIZONTAL
2794	<.S>	/d032/d032/d037/d219	BOX DRAWING LIGHT SHADE (25%)
2795	<:S>	/d032/d032/d037/d220	BOX DRAWING MEDIUM SHADE (50%)
2796	<?S>	/d032/d032/d037/d221	BOX DRAWING DARK SHADE (75%)
2797	<lB>	/d032/d032/d037/d222	BOX DRAWING SOLID LEFT HALF BLOCK
2798	<rB>	/d032/d032/d037/d223	BOX DRAWING SOLID RIGHT HALF BLOCK
2799	<cC>	/d032/d032/d037/d224	CLUB SYMBOL
2800	<cD>	/d032/d032/d037/d225	DIAMOND SYMBOL
2801	<Dr>	/d032/d032/d037/d226	BOX DRAWINGS DOWN HEAVY AND RIGHT LIGHT
2802	<Dl>	/d032/d032/d037/d227	BOX DRAWINGS DOWN HEAVY AND LEFT LIGHT

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2803	<Ur>	/d032/d032/d037/d228	BOX DRAWINGS UP HEAVY AND RIGHT LIGHT
2804		/d032/d032/d037/d229	BOX DRAWINGS UP HEAVY AND LEFT LIGHT
2805	<Vr>	/d032/d032/d037/d230	BOX DRAWINGS VERTICAL HEAVY AND RIGHT LIGHT
2806	<Vl>	/d032/d032/d037/d231	BOX DRAWINGS VERTICAL HEAVY AND LEFT LIGHT
2807	<dH>	/d032/d032/d037/d232	BOX DRAWINGS HORIZONTAL HEAVY AND DOWN LIGHT
2808	<uH>	/d032/d032/d037/d233	BOX DRAWINGS HORIZONTAL HEAVY AND UP LIGHT
2809	<vH>	/d032/d032/d037/d234	BOX DRAWINGS VERTICAL LIGHT AND HORIZONTAL HEAVY
2810	<Ob>	/d032/d032/d037/d235	CIRCLE BULLET EMPTY
2811	<Sb>	/d032/d032/d037/d236	CIRCLE BULLET SOLID
2812	<Sn>	/d032/d032/d037/d237	CIRCLE BULLET NEGATIVE
2813	<Pt>	/d032/d032/d037/d238	PESETA SYMBOL
2814	<NI>	/d032/d032/d037/d239	REVERSED NOT SIGN
2815	<cH>	/d032/d032/d037/d240	HEART SYMBOL
2816	<cS>	/d032/d032/d037/d241	SPADE SYMBOL
2817	<dR>	/d032/d032/d037/d242	BOX DRAWINGS DOWN LIGHT AND RIGHT HEAVY
2818	<dL>	/d032/d032/d037/d243	BOX DRAWINGS DOWN LIGHT AND LEFT HEAVY
2819	<uR>	/d032/d032/d037/d244	BOX DRAWINGS UP LIGHT AND RIGHT HEAVY
2820		/d032/d032/d037/d245	BOX DRAWINGS UP LIGHT AND LEFT HEAVY
2821	<vR>	/d032/d032/d037/d246	BOX DRAWINGS VERTICAL LIGHT AND RIGHT HEAVY
2822	<vL>	/d032/d032/d037/d247	BOX DRAWINGS VERTICAL LIGHT AND LEFT HEAVY
2823	<Dh>	/d032/d032/d037/d248	BOX DRAWINGS HORIZONTAL LIGHT AND DOWN HEAVY
2824	<Uh>	/d032/d032/d037/d249	BOX DRAWINGS HORIZONTAL LIGHT AND UP HEAVY
2825	<Vh>	/d032/d032/d037/d250	BOX DRAWINGS VERTICAL HEAVY AND HORIZONTAL LIGHT
2826	<Om>	/d032/d032/d037/d251	MEDIUM CIRCLE EMPTY
2827	<OM>	/d032/d032/d037/d252	MEDIUM CIRCLE SOLID
2828	<Ic>	/d032/d032/d037/d253	MEDIUM CIRCLE NEGATIVE
2829	<SM>	/d032/d032/d037/d254	SERVICE MARK SIGN
2830	<CG>	/d032/d032/d037/d255	CONGRUENCE SIGN
2831	<Ci>	/d032/d032/d038/d037	CIRCLE
2832	<(A>	/d032/d032/d038/d041	ARC SIGN
2833	</>V>	/d032/d032/d038/d046	RIGHTWARDS VECTOR ABOVE
2834	<!<>	/d032/d032/d038/d049	NOT LESS-THAN SIGN
2835	<<*>	/d032/d032/d038/d056	MUCH-LESS-THAN SIGN
2836	<!/>>	/d032/d032/d038/d065	NOT GREATER-THAN SIGN
2837	<*/>>	/d032/d032/d038/d072	MUCH-GREATER-THAN SIGN
2838	<<7>	/d032/d032/d038/d094	CEILING SIGN LEFT
2839	<7<>	/d032/d032/d038/d095	FLOOR SIGN LEFT
2840	</>7>	/d032/d032/d038/d110	CEILING SIGN RIGHT
2841	<7/>>	/d032/d032/d038/d111	FLOOR SIGN RIGHT
2842	<I2>	/d032/d032/d038/d121	DOUBLE INTEGRAL SIGN
2843	<0.>	/d032/d032/d038/d164	DOT IN RING
2844	<HI>	/d032/d032/d038/d177	HAS-AN-IMAGE SIGN
2845	<::>	/d032/d032/d038/d193	PROPORTION SIGN
2846	<FD>	/d032/d032/d038/d209	FORWARD DIAGONAL
2847	<LZ>	/d032/d032/d038/d223	LOZENGE
2848	<BD>	/d032/d032/d038/d225	BACKWARD DIAGONAL
2849	<1R>	/d032/d032/d039/d032	ROMAN NUMERAL ONE
2850	<2R>	/d032/d032/d039/d033	ROMAN NUMERAL TWO
2851	<3R>	/d032/d032/d039/d034	ROMAN NUMERAL THREE
2852	<4R>	/d032/d032/d039/d035	ROMAN NUMERAL FOUR
2853	<5R>	/d032/d032/d039/d036	ROMAN NUMERAL FIVE
2854	<6R>	/d032/d032/d039/d037	ROMAN NUMERAL SIX
2855	<7R>	/d032/d032/d039/d038	ROMAN NUMERAL SEVEN
2856	<8R>	/d032/d032/d039/d039	ROMAN NUMERAL EIGHT
2857	<9R>	/d032/d032/d039/d040	ROMAN NUMERAL NINE
2858	<aR>	/d032/d032/d039/d041	ROMAN NUMERAL TEN
2859	 	/d032/d032/d039/d042	ROMAN NUMERAL ELEVEN

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2860	<cR>	/d032/d032/d039/d043	ROMAN NUMERAL TWELVE
2861	<IO>	/d032/d032/d040/d161	CYRILLIC CAPITAL LETTER IO
2862	<D%>	/d032/d032/d040/d162	CYRILLIC CAPITAL LETTER DJE (Serbocroatian)
2863	<G%>	/d032/d032/d040/d163	CYRILLIC CAPITAL LETTER GJE (Macedonian)
2864	<IE>	/d032/d032/d040/d164	CYRILLIC CAPITAL LETTER UKRAINIAN IE
2865	<DS>	/d032/d032/d040/d165	CYRILLIC CAPITAL LETTER DZE (Macedonian)
2866	<II>	/d032/d032/d040/d166	CYRILLIC CAPITAL LETTER BYELORUSSIAN-UKRAINIAN I
2867	<YI>	/d032/d032/d040/d167	CYRILLIC CAPITAL LETTER YI (Ukrainian)
2868	<J%>	/d032/d032/d040/d168	CYRILLIC CAPITAL LETTER JE
2869	<LJ>	/d032/d032/d040/d169	CYRILLIC CAPITAL LETTER LJE
2870	<NJ>	/d032/d032/d040/d170	CYRILLIC CAPITAL LETTER NJE
2871	<Ts>	/d032/d032/d040/d171	CYRILLIC CAPITAL LETTER TSHE (Serbocroatian)
2872	<KJ>	/d032/d032/d040/d172	CYRILLIC CAPITAL LETTER KJE (Macedonian)
2873	<V%>	/d032/d032/d040/d174	CYRILLIC CAPITAL LETTER SHORT U (Byelorussian)
2874	<DZ>	/d032/d032/d040/d175	CYRILLIC CAPITAL LETTER DZHE
2875	<A=>	/d032/d032/d040/d176	CYRILLIC CAPITAL LETTER A
2876	<B=>	/d032/d032/d040/d177	CYRILLIC CAPITAL LETTER BE
2877	<V=>	/d032/d032/d040/d178	CYRILLIC CAPITAL LETTER VE
2878	<G=>	/d032/d032/d040/d179	CYRILLIC CAPITAL LETTER GHE
2879	<D=>	/d032/d032/d040/d180	CYRILLIC CAPITAL LETTER DE
2880	<E=>	/d032/d032/d040/d181	CYRILLIC CAPITAL LETTER IE
2881	<Z%>	/d032/d032/d040/d182	CYRILLIC CAPITAL LETTER ZHE
2882	<Z=>	/d032/d032/d040/d183	CYRILLIC CAPITAL LETTER ZE
2883	<I=>	/d032/d032/d040/d184	CYRILLIC CAPITAL LETTER I
2884	<J=>	/d032/d032/d040/d185	CYRILLIC CAPITAL LETTER SHORT I
2885	<K=>	/d032/d032/d040/d186	CYRILLIC CAPITAL LETTER KA
2886	<L=>	/d032/d032/d040/d187	CYRILLIC CAPITAL LETTER EL
2887	<M=>	/d032/d032/d040/d188	CYRILLIC CAPITAL LETTER EM
2888	<N=>	/d032/d032/d040/d189	CYRILLIC CAPITAL LETTER EN
2889	<O=>	/d032/d032/d040/d190	CYRILLIC CAPITAL LETTER O
2890	<P=>	/d032/d032/d040/d191	CYRILLIC CAPITAL LETTER PE
2891	<R=>	/d032/d032/d040/d192	CYRILLIC CAPITAL LETTER ER
2892	<S=>	/d032/d032/d040/d193	CYRILLIC CAPITAL LETTER ES
2893	<T=>	/d032/d032/d040/d194	CYRILLIC CAPITAL LETTER TE
2894	<U=>	/d032/d032/d040/d195	CYRILLIC CAPITAL LETTER U
2895	<F=>	/d032/d032/d040/d196	CYRILLIC CAPITAL LETTER EF
2896	<H=>	/d032/d032/d040/d197	CYRILLIC CAPITAL LETTER HA
2897	<C=>	/d032/d032/d040/d198	CYRILLIC CAPITAL LETTER TSE
2898	<C%>	/d032/d032/d040/d199	CYRILLIC CAPITAL LETTER CHE
2899	<S%>	/d032/d032/d040/d200	CYRILLIC CAPITAL LETTER SHA
2900	<Sc>	/d032/d032/d040/d201	CYRILLIC CAPITAL LETTER SHCHA
2901	<=">	/d032/d032/d040/d202	CYRILLIC CAPITAL HARD SIGN
2902	<Y=>	/d032/d032/d040/d203	CYRILLIC CAPITAL LETTER YERU
2903	<% ">	/d032/d032/d040/d204	CYRILLIC CAPITAL SOFT SIGN
2904	<JE>	/d032/d032/d040/d205	CYRILLIC CAPITAL LETTER E
2905	<JU>	/d032/d032/d040/d206	CYRILLIC CAPITAL LETTER YU
2906	<JA>	/d032/d032/d040/d207	CYRILLIC CAPITAL LETTER YA
2907	<a=>	/d032/d032/d040/d208	CYRILLIC SMALL LETTER A
2908	<b=>	/d032/d032/d040/d209	CYRILLIC SMALL LETTER BE
2909	<v=>	/d032/d032/d040/d210	CYRILLIC SMALL LETTER VE
2910	<g=>	/d032/d032/d040/d211	CYRILLIC SMALL LETTER GHE
2911	<d=>	/d032/d032/d040/d212	CYRILLIC SMALL LETTER DE
2912	<e=>	/d032/d032/d040/d213	CYRILLIC SMALL LETTER IE
2913	<z%>	/d032/d032/d040/d214	CYRILLIC SMALL LETTER ZHE
2914	<z=>	/d032/d032/d040/d215	CYRILLIC SMALL LETTER ZE
2915	<i=>	/d032/d032/d040/d216	CYRILLIC SMALL LETTER I
2916	<j=>	/d032/d032/d040/d217	CYRILLIC SMALL LETTER SHORT I

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2917	<k=>	/d032/d032/d040/d218	CYRILLIC SMALL LETTER KA
2918	<l=>	/d032/d032/d040/d219	CYRILLIC SMALL LETTER EL
2919	<m=>	/d032/d032/d040/d220	CYRILLIC SMALL LETTER EM
2920	<n=>	/d032/d032/d040/d221	CYRILLIC SMALL LETTER EN
2921	<o=>	/d032/d032/d040/d222	CYRILLIC SMALL LETTER O
2922	<p=>	/d032/d032/d040/d223	CYRILLIC SMALL LETTER PE
2923	<r=>	/d032/d032/d040/d224	CYRILLIC SMALL LETTER ER
2924	<s=>	/d032/d032/d040/d225	CYRILLIC SMALL LETTER ES
2925	<t=>	/d032/d032/d040/d226	CYRILLIC SMALL LETTER TE
2926	<u=>	/d032/d032/d040/d227	CYRILLIC SMALL LETTER U
2927	<f=>	/d032/d032/d040/d228	CYRILLIC SMALL LETTER EF
2928	<h=>	/d032/d032/d040/d229	CYRILLIC SMALL LETTER HA
2929	<c=>	/d032/d032/d040/d230	CYRILLIC SMALL LETTER TSE
2930	<c%>	/d032/d032/d040/d231	CYRILLIC SMALL LETTER CHE
2931	<s%>	/d032/d032/d040/d232	CYRILLIC SMALL LETTER SHA
2932	<sc>	/d032/d032/d040/d233	CYRILLIC SMALL LETTER SHCHA
2933	<='>	/d032/d032/d040/d234	CYRILLIC SMALL HARD SIGN
2934	<y=>	/d032/d032/d040/d235	CYRILLIC SMALL LETTER YERU
2935	<% '>	/d032/d032/d040/d236	CYRILLIC SMALL SOFT SIGN
2936	<je>	/d032/d032/d040/d237	CYRILLIC SMALL LETTER E
2937	<ju>	/d032/d032/d040/d238	CYRILLIC SMALL LETTER YU
2938	<ja>	/d032/d032/d040/d239	CYRILLIC SMALL LETTER YA
2939	<N0>	/d032/d032/d040/d240	NUMERO SIGN
2940	<io>	/d032/d032/d040/d241	CYRILLIC SMALL LETTER IO
2941	<d%>	/d032/d032/d040/d242	CYRILLIC SMALL LETTER DJE (Serbocroatian)
2942	<g%>	/d032/d032/d040/d243	CYRILLIC SMALL LETTER GJE (Macedonian)
2943	<ie>	/d032/d032/d040/d244	CYRILLIC SMALL LETTER UKRAINIAN IE
2944	<ds>	/d032/d032/d040/d245	CYRILLIC SMALL LETTER DZE (Macedonian)
2945	<ii>	/d032/d032/d040/d246	CYRILLIC SMALL LETTER BYELORUSSIAN-UKRAINIAN I
2946	<yi>	/d032/d032/d040/d247	CYRILLIC SMALL LETTER YI (Ukrainian)
2947	<j%>	/d032/d032/d040/d248	CYRILLIC SMALL LETTER JE
2948	<l j>	/d032/d032/d040/d249	CYRILLIC SMALL LETTER LJE
2949	<nj>	/d032/d032/d040/d250	CYRILLIC SMALL LETTER NJE
2950	<ts>	/d032/d032/d040/d251	CYRILLIC SMALL LETTER TSHE (Serbocroatian)
2951	<kj>	/d032/d032/d040/d252	CYRILLIC SMALL LETTER KJE (Macedonian)
2952	<v%>	/d032/d032/d040/d254	CYRILLIC SMALL LETTER SHORT U (Byelorussian)
2953	<dz>	/d032/d032/d040/d255	CYRILLIC SMALL LETTER DZHE
2954	<i3>	/d032/d032/d042/d160	GREEK IOTA BELOW
2955	< ; >	/d032/d032/d042/d161	GREEK DAISA PNEUMATA (rough)
2956	< , , >	/d032/d032/d042/d162	GREEK PSILI PNEUMATA (smooth)
2957	<!*>	/d032/d032/d042/d164	GREEK VARIA
2958	<?*>	/d032/d032/d042/d165	GREEK PERISPOMENI
2959	< ; ' >	/d032/d032/d042/d166	GREEK DAISA AND ACUTE ACCENT
2960	< , ' >	/d032/d032/d042/d167	GREEK PSILI AND ACUTE ACCENT
2961	< ; ! >	/d032/d032/d042/d168	GREEK DAISA AND VARIA
2962	< , ! >	/d032/d032/d042/d169	GREEK PSILI AND VARIA
2963	< ? ; >	/d032/d032/d042/d170	GREEK PERISPOMENI AND DAISA
2964	< ? , >	/d032/d032/d042/d171	GREEK PERISPOMENI AND PSILI
2965	< ! ; >	/d032/d032/d042/d174	GREEK VARIA AND DIAERESIS
2966	< ? ; >	/d032/d032/d042/d175	GREEK PERISPOMENI AND DIAERESIS
2967	<I3>	/d032/d032/d042/d176	GREEK CAPITAL LETTER IOTA WITH PERISPOMENI
2968	#		AND PSILI
2969	<' %>	/d032/d032/d042/d181	ACUTE ACCENT AND DIAERESIS (Tonos and Dialytica)
2970	<A%>	/d032/d032/d042/d182	GREEK CAPITAL LETTER ALPHA WITH ACUTE
2971	<E%>	/d032/d032/d042/d184	GREEK CAPITAL LETTER EPSILON WITH ACUTE
2972	<Y%>	/d032/d032/d042/d185	GREEK CAPITAL LETTER ETA WITH ACUTE
2973	<I%>	/d032/d032/d042/d186	GREEK CAPITAL LETTER IOTA WITH ACUTE

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

2974	<O%>	/d032/d032/d042/d188	GREEK CAPITAL LETTER OMICRON WITH ACUTE
2975	<U%>	/d032/d032/d042/d190	GREEK CAPITAL LETTER UPSILON WITH ACUTE
2976	<W%>	/d032/d032/d042/d191	GREEK CAPITAL LETTER OMEGA WITH ACUTE
2977	<A*>	/d032/d032/d042/d193	GREEK CAPITAL LETTER ALPHA
2978	<B*>	/d032/d032/d042/d194	GREEK CAPITAL LETTER BETA
2979	<G*>	/d032/d032/d042/d195	GREEK CAPITAL LETTER GAMMA
2980	<D*>	/d032/d032/d042/d196	GREEK CAPITAL LETTER DELTA
2981	<E*>	/d032/d032/d042/d197	GREEK CAPITAL LETTER EPSILON
2982	<Z*>	/d032/d032/d042/d198	GREEK CAPITAL LETTER ZETA
2983	<Y*>	/d032/d032/d042/d199	GREEK CAPITAL LETTER ETA
2984	<H*>	/d032/d032/d042/d200	GREEK CAPITAL LETTER THETA
2985	<I*>	/d032/d032/d042/d201	GREEK CAPITAL LETTER IOTA
2986	<K*>	/d032/d032/d042/d202	GREEK CAPITAL LETTER KAPPA
2987	<L*>	/d032/d032/d042/d203	GREEK CAPITAL LETTER LAMDA
2988	<M*>	/d032/d032/d042/d204	GREEK CAPITAL LETTER MU
2989	<N*>	/d032/d032/d042/d205	GREEK CAPITAL LETTER NU
2990	<C*>	/d032/d032/d042/d206	GREEK CAPITAL LETTER XI
2991	<O*>	/d032/d032/d042/d207	GREEK CAPITAL LETTER OMICRON
2992	<P*>	/d032/d032/d042/d208	GREEK CAPITAL LETTER PI
2993	<R*>	/d032/d032/d042/d209	GREEK CAPITAL LETTER RHO
2994	<S*>	/d032/d032/d042/d211	GREEK CAPITAL LETTER SIGMA
2995	<T*>	/d032/d032/d042/d212	GREEK CAPITAL LETTER TAU
2996	<U*>	/d032/d032/d042/d213	GREEK CAPITAL LETTER UPSILON
2997	<F*>	/d032/d032/d042/d214	GREEK CAPITAL LETTER PHI
2998	<X*>	/d032/d032/d042/d215	GREEK CAPITAL LETTER CHI
2999	<Q*>	/d032/d032/d042/d216	GREEK CAPITAL LETTER PSI
3000	<W*>	/d032/d032/d042/d217	GREEK CAPITAL LETTER OMEGA
3001	<J*>	/d032/d032/d042/d218	GREEK CAPITAL LETTER IOTA WITH DIAERESIS
3002	<V*>	/d032/d032/d042/d219	GREEK CAPITAL LETTER UPSILON WITH DIAERESIS
3003	<a%>	/d032/d032/d042/d220	GREEK SMALL LETTER ALPHA WITH ACUTE
3004	<e%>	/d032/d032/d042/d221	GREEK SMALL LETTER EPSILON WITH ACUTE
3005	<y%>	/d032/d032/d042/d222	GREEK SMALL LETTER ETA WITH ACUTE
3006	<i%>	/d032/d032/d042/d223	GREEK SMALL LETTER IOTA WITH ACUTE
3007	<a*>	/d032/d032/d042/d225	GREEK SMALL LETTER ALPHA
3008	<b*>	/d032/d032/d042/d226	GREEK SMALL LETTER BETA
3009	<g*>	/d032/d032/d042/d227	GREEK SMALL LETTER GAMMA
3010	<d*>	/d032/d032/d042/d228	GREEK SMALL LETTER DELTA
3011	<e*>	/d032/d032/d042/d229	GREEK SMALL LETTER EPSILON
3012	<z*>	/d032/d032/d042/d230	GREEK SMALL LETTER ZETA
3013	<y*>	/d032/d032/d042/d231	GREEK SMALL LETTER ETA
3014	<h*>	/d032/d032/d042/d232	GREEK SMALL LETTER THETA
3015	<i*>	/d032/d032/d042/d233	GREEK SMALL LETTER IOTA
3016	<k*>	/d032/d032/d042/d234	GREEK SMALL LETTER KAPPA
3017	<l*>	/d032/d032/d042/d235	GREEK SMALL LETTER LAMDA
3018	<m*>	/d032/d032/d042/d236	GREEK SMALL LETTER MU
3019	<n*>	/d032/d032/d042/d237	GREEK SMALL LETTER NU
3020	<c*>	/d032/d032/d042/d238	GREEK SMALL LETTER XI
3021	<o*>	/d032/d032/d042/d239	GREEK SMALL LETTER OMICRON
3022	<p*>	/d032/d032/d042/d240	GREEK SMALL LETTER PI
3023	<r*>	/d032/d032/d042/d241	GREEK SMALL LETTER RHO
3024	<s*>	/d032/d032/d042/d242	GREEK SMALL LETTER FINAL SIGMA
3025	<s*>	/d032/d032/d042/d243	GREEK SMALL LETTER SIGMA
3026	<t*>	/d032/d032/d042/d244	GREEK SMALL LETTER TAU
3027	<u*>	/d032/d032/d042/d245	GREEK SMALL LETTER UPSILON
3028	<f*>	/d032/d032/d042/d246	GREEK SMALL LETTER PHI
3029	<x*>	/d032/d032/d042/d247	GREEK SMALL LETTER CHI
3030	<q*>	/d032/d032/d042/d248	GREEK SMALL LETTER PSI

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

3031	<w*>	/d032/d032/d042/d249	GREEK SMALL LETTER OMEGA
3032	<j*>	/d032/d032/d042/d250	GREEK SMALL LETTER IOTA WITH DIAERESIS
3033	<v*>	/d032/d032/d042/d251	GREEK SMALL LETTER UPSILON WITH DIAERESIS
3034	<o%>	/d032/d032/d042/d252	GREEK SMALL LETTER OMICRON WITH ACUTE
3035	<u%>	/d032/d032/d042/d253	GREEK SMALL LETTER UPSILON WITH ACUTE
3036	<w%>	/d032/d032/d042/d254	GREEK SMALL LETTER OMEGA WITH ACUTE
3037	<p+>	/d032/d032/d044/d035	ARABIC LETTER PEH
3038	<v+>	/d032/d032/d044/d040	ARABIC LETTER VEH
3039	<gf>	/d032/d032/d044/d052	ARABIC LETTER GAF
3040	<, +>	/d032/d032/d044/d172	ARABIC COMMA
3041	<; +>	/d032/d032/d044/d187	ARABIC SEMICOLON
3042	<? +>	/d032/d032/d044/d191	ARABIC QUESTION MARK
3043	<H'>	/d032/d032/d044/d193	ARABIC LETTER HAMZA
3044	<aM>	/d032/d032/d044/d194	ARABIC LETTER ALEF WITH MADDA ABOVE
3045	<aH>	/d032/d032/d044/d195	ARABIC LETTER ALEF WITH HAMZA ABOVE
3046	<wH>	/d032/d032/d044/d196	ARABIC LETTER WAW WITH HAMZA ABOVE
3047	<ah>	/d032/d032/d044/d197	ARABIC LETTER ALEF WITH HAMZA BELOW
3048	<yH>	/d032/d032/d044/d198	ARABIC LETTER YEH WITH HAMZA ABOVE
3049	<a+>	/d032/d032/d044/d199	ARABIC LETTER ALEF
3050	<b+>	/d032/d032/d044/d200	ARABIC LETTER BEH
3051	<tm>	/d032/d032/d044/d201	ARABIC LETTER TEH MARBUTA
3052	<t+>	/d032/d032/d044/d202	ARABIC LETTER TEH
3053	<tk>	/d032/d032/d044/d203	ARABIC LETTER THEH
3054	<g+>	/d032/d032/d044/d204	ARABIC LETTER JEEM
3055	<hk>	/d032/d032/d044/d205	ARABIC LETTER HAH
3056	<x+>	/d032/d032/d044/d206	ARABIC LETTER KHAH
3057	<d+>	/d032/d032/d044/d207	ARABIC LETTER DAL
3058	<dk>	/d032/d032/d044/d208	ARABIC LETTER THAL
3059	<r+>	/d032/d032/d044/d209	ARABIC LETTER RA
3060	<z+>	/d032/d032/d044/d210	ARABIC LETTER ZAIN
3061	<s+>	/d032/d032/d044/d211	ARABIC LETTER SEEN
3062	<sn>	/d032/d032/d044/d212	ARABIC LETTER SHEEN
3063	<c+>	/d032/d032/d044/d213	ARABIC LETTER SAD
3064	<dd>	/d032/d032/d044/d214	ARABIC LETTER DAD
3065	<tj>	/d032/d032/d044/d215	ARABIC LETTER TAH
3066	<zH>	/d032/d032/d044/d216	ARABIC LETTER ZAH
3067	<e+>	/d032/d032/d044/d217	ARABIC LETTER AIN
3068	<i+>	/d032/d032/d044/d218	ARABIC LETTER GHAIN
3069	<++>	/d032/d032/d044/d224	ARABIC TATWEEL
3070	<f+>	/d032/d032/d044/d225	ARABIC LETTER FEH
3071	<q+>	/d032/d032/d044/d226	ARABIC LETTER QAF
3072	<k+>	/d032/d032/d044/d227	ARABIC LETTER KAF
3073	<l+>	/d032/d032/d044/d228	ARABIC LETTER LAM
3074	<m+>	/d032/d032/d044/d229	ARABIC LETTER MEEM
3075	<n+>	/d032/d032/d044/d230	ARABIC LETTER NOON
3076	<h+>	/d032/d032/d044/d231	ARABIC LETTER HEH
3077	<w+>	/d032/d032/d044/d232	ARABIC LETTER WAW
3078	<j+>	/d032/d032/d044/d233	ARABIC LETTER ALEF MAKSURA
3079	<y+>	/d032/d032/d044/d234	ARABIC LETTER YEH
3080	<: +>	/d032/d032/d044/d235	ARABIC FATHATAN
3081	<" +>	/d032/d032/d044/d236	ARABIC DAMMATAN
3082	<= +>	/d032/d032/d044/d237	ARABIC KASRATAN
3083	</ +>	/d032/d032/d044/d238	ARABIC FATHA
3084	<' +>	/d032/d032/d044/d239	ARABIC DAMMA
3085	<1+>	/d032/d032/d044/d240	ARABIC KASRA
3086	<3+>	/d032/d032/d044/d241	ARABIC SHADDA
3087	<0+>	/d032/d032/d044/d242	ARABIC SUKUN

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

3088	<A+>	/d032/d032/d045/d224	HEBREW LETTER ALEF
3089	<B+>	/d032/d032/d045/d225	HEBREW LETTER BET
3090	<G+>	/d032/d032/d045/d226	HEBREW LETTER GIMEL
3091	<D+>	/d032/d032/d045/d227	HEBREW LETTER DALET
3092	<H+>	/d032/d032/d045/d228	HEBREW LETTER HE
3093	<W+>	/d032/d032/d045/d229	HEBREW LETTER VAV
3094	<Z+>	/d032/d032/d045/d230	HEBREW LETTER ZAYIN
3095	<X+>	/d032/d032/d045/d231	HEBREW LETTER HET
3096	<Tj>	/d032/d032/d045/d232	HEBREW LETTER TET
3097	<J+>	/d032/d032/d045/d233	HEBREW LETTER YOD
3098	<K%>	/d032/d032/d045/d234	HEBREW LETTER FINAL KAF
3099	<K+>	/d032/d032/d045/d235	HEBREW LETTER KAF
3100	<L+>	/d032/d032/d045/d236	HEBREW LETTER LAMED
3101	<M%>	/d032/d032/d045/d237	HEBREW LETTER FINAL MEM
3102	<M+>	/d032/d032/d045/d238	HEBREW LETTER MEM
3103	<N%>	/d032/d032/d045/d239	HEBREW LETTER FINAL NUN
3104	<N+>	/d032/d032/d045/d240	HEBREW LETTER NUN
3105	<S+>	/d032/d032/d045/d241	HEBREW LETTER SAMEKH
3106	<E+>	/d032/d032/d045/d242	HEBREW LETTER AYIN
3107	<P%>	/d032/d032/d045/d243	HEBREW LETTER FINAL PE
3108	<P+>	/d032/d032/d045/d244	HEBREW LETTER PE
3109	<Zj>	/d032/d032/d045/d245	HEBREW LETTER FINAL TSADI
3110	<ZJ>	/d032/d032/d045/d246	HEBREW LETTER TSADI
3111	<Q+>	/d032/d032/d045/d247	HEBREW LETTER QOF
3112	<R+>	/d032/d032/d045/d248	HEBREW LETTER RESH
3113	<Sh>	/d032/d032/d045/d249	HEBREW LETTER SIN
3114	<T+>	/d032/d032/d045/d250	HEBREW LETTER TAV
3115	<IS>	/d032/d032/d046/d032	IDEOGRAPHIC SPACE
3116	<,>	/d032/d032/d046/d033	IDEOGRAPHIC COMMA
3117	<.>	/d032/d032/d046/d034	IDEOGRAPHIC FULL STOP
3118	<+>	/d032/d032/d046/d035	DITTO MARK
3119	<+>	/d032/d032/d046/d036	IDEOGRAPHIC DITTO MARK
3120	<*>	/d032/d032/d046/d037	IDEOGRAPHIC REPETITION MARK
3121	<:>	/d032/d032/d046/d038	IDEOGRAPHIC CLOSING MARK
3122	<0>	/d032/d032/d046/d039	IDEOGRAPHIC NUMBER ZERO
3123	<<+>	/d032/d032/d046/d042	LEFT-POINTING DOUBLE ANGLE BRACKET
3124	</>+>	/d032/d032/d046/d043	RIGHT-POINTING DOUBLE ANGLE BRACKET
3125	<<'>	/d032/d032/d046/d044	IDEOGRAPHIC LEFT BRACKET
3126	</>'>	/d032/d032/d046/d045	IDEOGRAPHIC RIGHT BRACKET
3127	<<">	/d032/d032/d046/d046	IDEOGRAPHIC LEFT DOUBLE BRACKET
3128	</>">	/d032/d032/d046/d047	IDEOGRAPHIC RIGHT DOUBLE BRACKET
3129	<('>	/d032/d032/d046/d048	LEFT BOLDFACE SQUARE BRACKET
3130	<)>	/d032/d032/d046/d049	RIGHT BOLDFACE SQUARE BRACKET
3131	<=/>	/d032/d032/d046/d050	POSTAL MARK
3132	<=>	/d032/d032/d046/d051	GETA MARK
3133	<('>	/d032/d032/d046/d052	LEFT TORTOISE-SHELL BRACKET
3134	<)'>	/d032/d032/d046/d053	RIGHT TORTOISE-SHELL BRACKET
3135	<KM>	/d032/d032/d046/d054	KOME MARK
3136	<b4>	/d032/d032/d046/d069	BOPOMOFO LETTER B
3137	<p4>	/d032/d032/d046/d070	BOPOMOFO LETTER P
3138	<m4>	/d032/d032/d046/d071	BOPOMOFO LETTER M
3139	<f4>	/d032/d032/d046/d072	BOPOMOFO LETTER F
3140	<d4>	/d032/d032/d046/d073	BOPOMOFO LETTER D
3141	<t4>	/d032/d032/d046/d074	BOPOMOFO LETTER T
3142	<n4>	/d032/d032/d046/d075	BOPOMOFO LETTER N
3143	<l4>	/d032/d032/d046/d076	BOPOMOFO LETTER L
3144	<g4>	/d032/d032/d046/d077	BOPOMOFO LETTER G

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

3145	<k4>	/d032/d032/d046/d078	BOPOMOFO LETTER K
3146	<h4>	/d032/d032/d046/d079	BOPOMOFO LETTER H
3147	<j4>	/d032/d032/d046/d080	BOPOMOFO LETTER J
3148	<q4>	/d032/d032/d046/d081	BOPOMOFO LETTER Q
3149	<x4>	/d032/d032/d046/d082	BOPOMOFO LETTER X
3150	<zh>	/d032/d032/d046/d083	BOPOMOFO LETTER ZH
3151	<ch>	/d032/d032/d046/d084	BOPOMOFO LETTER CH
3152	<sh>	/d032/d032/d046/d085	BOPOMOFO LETTER SH
3153	<r4>	/d032/d032/d046/d086	BOPOMOFO LETTER R
3154	<z4>	/d032/d032/d046/d087	BOPOMOFO LETTER Z
3155	<c4>	/d032/d032/d046/d088	BOPOMOFO LETTER C
3156	<s4>	/d032/d032/d046/d089	BOPOMOFO LETTER S
3157	<a4>	/d032/d032/d046/d090	BOPOMOFO LETTER A
3158	<o4>	/d032/d032/d046/d091	BOPOMOFO LETTER O
3159	<e4>	/d032/d032/d046/d092	BOPOMOFO LETTER E
3160	<eh>	/d032/d032/d046/d093	BOPOMOFO LETTER EH
3161	<ai>	/d032/d032/d046/d094	BOPOMOFO LETTER AI
3162	<ei>	/d032/d032/d046/d095	BOPOMOFO LETTER EI
3163	<au>	/d032/d032/d046/d096	BOPOMOFO LETTER AU
3164	<ou>	/d032/d032/d046/d097	BOPOMOFO LETTER OU
3165	<an>	/d032/d032/d046/d098	BOPOMOFO LETTER AN
3166	<en>	/d032/d032/d046/d099	BOPOMOFO LETTER EN
3167	<aN>	/d032/d032/d046/d100	BOPOMOFO LETTER ANG
3168	<eN>	/d032/d032/d046/d101	BOPOMOFO LETTER ENG
3169	<er>	/d032/d032/d046/d102	BOPOMOFO LETTER ER
3170	<i4>	/d032/d032/d046/d103	BOPOMOFO LETTER I
3171	<u4>	/d032/d032/d046/d104	BOPOMOFO LETTER U
3172	<iu>	/d032/d032/d046/d105	BOPOMOFO LETTER IU
3173	<A5>	/d032/d032/d047/d033	HIRAGANA LETTER SMALL A
3174	<a5>	/d032/d032/d047/d034	HIRAGANA LETTER A
3175	<I5>	/d032/d032/d047/d035	HIRAGANA LETTER SMALL I
3176	<i5>	/d032/d032/d047/d036	HIRAGANA LETTER I
3177	<U5>	/d032/d032/d047/d037	HIRAGANA LETTER SMALL U
3178	<u5>	/d032/d032/d047/d038	HIRAGANA LETTER U
3179	<E5>	/d032/d032/d047/d039	HIRAGANA LETTER SMALL E
3180	<e5>	/d032/d032/d047/d040	HIRAGANA LETTER E
3181	<O5>	/d032/d032/d047/d041	HIRAGANA LETTER SMALL O
3182	<o5>	/d032/d032/d047/d042	HIRAGANA LETTER O
3183	<ka>	/d032/d032/d047/d043	HIRAGANA LETTER KA
3184	<ga>	/d032/d032/d047/d044	HIRAGANA LETTER GA
3185	<ki>	/d032/d032/d047/d045	HIRAGANA LETTER KI
3186	<gi>	/d032/d032/d047/d046	HIRAGANA LETTER GI
3187	<ku>	/d032/d032/d047/d047	HIRAGANA LETTER KU
3188	<gu>	/d032/d032/d047/d048	HIRAGANA LETTER GU
3189	<ke>	/d032/d032/d047/d049	HIRAGANA LETTER KE
3190	<ge>	/d032/d032/d047/d050	HIRAGANA LETTER GE
3191	<ko>	/d032/d032/d047/d051	HIRAGANA LETTER KO
3192	<go>	/d032/d032/d047/d052	HIRAGANA LETTER GO
3193	<sa>	/d032/d032/d047/d053	HIRAGANA LETTER SA
3194	<za>	/d032/d032/d047/d054	HIRAGANA LETTER ZA
3195	<si>	/d032/d032/d047/d055	HIRAGANA LETTER SI
3196	<zi>	/d032/d032/d047/d056	HIRAGANA LETTER ZI
3197	<su>	/d032/d032/d047/d057	HIRAGANA LETTER SU
3198	<zu>	/d032/d032/d047/d058	HIRAGANA LETTER ZU
3199	<se>	/d032/d032/d047/d059	HIRAGANA LETTER SE
3200	<ze>	/d032/d032/d047/d060	HIRAGANA LETTER ZE
3201	<so>	/d032/d032/d047/d061	HIRAGANA LETTER SO

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

3202	<zo>	/d032/d032/d047/d062	HIRAGANA LETTER ZO
3203	<ta>	/d032/d032/d047/d063	HIRAGANA LETTER TA
3204	<da>	/d032/d032/d047/d064	HIRAGANA LETTER DA
3205	<ti>	/d032/d032/d047/d065	HIRAGANA LETTER TI
3206	<di>	/d032/d032/d047/d066	HIRAGANA LETTER DI
3207	<tU>	/d032/d032/d047/d067	HIRAGANA LETTER SMALL TU
3208	<tu>	/d032/d032/d047/d068	HIRAGANA LETTER TU
3209	<du>	/d032/d032/d047/d069	HIRAGANA LETTER DU
3210	<te>	/d032/d032/d047/d070	HIRAGANA LETTER TE
3211	<de>	/d032/d032/d047/d071	HIRAGANA LETTER DE
3212	<to>	/d032/d032/d047/d072	HIRAGANA LETTER TO
3213	<do>	/d032/d032/d047/d073	HIRAGANA LETTER DO
3214	<na>	/d032/d032/d047/d074	HIRAGANA LETTER NA
3215	<ni>	/d032/d032/d047/d075	HIRAGANA LETTER NI
3216	<nu>	/d032/d032/d047/d076	HIRAGANA LETTER NU
3217	<ne>	/d032/d032/d047/d077	HIRAGANA LETTER NE
3218	<no>	/d032/d032/d047/d078	HIRAGANA LETTER NO
3219	<ha>	/d032/d032/d047/d079	HIRAGANA LETTER HA
3220	<ba>	/d032/d032/d047/d080	HIRAGANA LETTER BA
3221	<pa>	/d032/d032/d047/d081	HIRAGANA LETTER PA
3222	<hi>	/d032/d032/d047/d082	HIRAGANA LETTER HI
3223	<bi>	/d032/d032/d047/d083	HIRAGANA LETTER BI
3224	<pi>	/d032/d032/d047/d084	HIRAGANA LETTER PI
3225	<hu>	/d032/d032/d047/d085	HIRAGANA LETTER HU
3226	<bu>	/d032/d032/d047/d086	HIRAGANA LETTER BU
3227	<pu>	/d032/d032/d047/d087	HIRAGANA LETTER PU
3228	<he>	/d032/d032/d047/d088	HIRAGANA LETTER HE
3229	<be>	/d032/d032/d047/d089	HIRAGANA LETTER BE
3230	<pe>	/d032/d032/d047/d090	HIRAGANA LETTER PE
3231	<ho>	/d032/d032/d047/d091	HIRAGANA LETTER HO
3232	<bo>	/d032/d032/d047/d092	HIRAGANA LETTER BO
3233	<po>	/d032/d032/d047/d093	HIRAGANA LETTER PO
3234	<ma>	/d032/d032/d047/d094	HIRAGANA LETTER MA
3235	<mi>	/d032/d032/d047/d095	HIRAGANA LETTER MI
3236	<mu>	/d032/d032/d047/d096	HIRAGANA LETTER MU
3237	<me>	/d032/d032/d047/d097	HIRAGANA LETTER ME
3238	<mo>	/d032/d032/d047/d098	HIRAGANA LETTER MO
3239	<yA>	/d032/d032/d047/d099	HIRAGANA LETTER SMALL YA
3240	<ya>	/d032/d032/d047/d100	HIRAGANA LETTER YA
3241	<yU>	/d032/d032/d047/d101	HIRAGANA LETTER SMALL YU
3242	<yu>	/d032/d032/d047/d102	HIRAGANA LETTER YU
3243	<yO>	/d032/d032/d047/d103	HIRAGANA LETTER SMALL YO
3244	<yo>	/d032/d032/d047/d104	HIRAGANA LETTER YO
3245	<ra>	/d032/d032/d047/d105	HIRAGANA LETTER RA
3246	<ri>	/d032/d032/d047/d106	HIRAGANA LETTER RI
3247	<ru>	/d032/d032/d047/d107	HIRAGANA LETTER RU
3248	<re>	/d032/d032/d047/d108	HIRAGANA LETTER RE
3249	<ro>	/d032/d032/d047/d109	HIRAGANA LETTER RO
3250	<wA>	/d032/d032/d047/d110	HIRAGANA LETTER SMALL WA
3251	<wa>	/d032/d032/d047/d111	HIRAGANA LETTER WA
3252	<wi>	/d032/d032/d047/d112	HIRAGANA LETTER WI
3253	<we>	/d032/d032/d047/d113	HIRAGANA LETTER WE
3254	<wo>	/d032/d032/d047/d114	HIRAGANA LETTER WO
3255	<n5>	/d032/d032/d047/d115	HIRAGANA LETTER N
3256	<"5>	/d032/d032/d047/d122	HIRAGANA-KATAKANA VOICED SOUND MARK
3257	<05>	/d032/d032/d047/d123	HIRAGANA-KATAKANA SEMI-VOICED SOUND MARK
3258	<*5>	/d032/d032/d047/d124	HIRAGANA ITERATION MARK

3259	<+5>	/d032/d032/d047/d125	HIRAGANA VOICED ITERATION MARK
3260	<a6>	/d032/d032/d047/d161	KATAKANA LETTER SMALL A
3261	<A6>	/d032/d032/d047/d162	KATAKANA LETTER A
3262	<i6>	/d032/d032/d047/d163	KATAKANA LETTER SMALL I
3263	<I6>	/d032/d032/d047/d164	KATAKANA LETTER I
3264	<u6>	/d032/d032/d047/d165	KATAKANA LETTER SMALL U
3265	<U6>	/d032/d032/d047/d166	KATAKANA LETTER U
3266	<e6>	/d032/d032/d047/d167	KATAKANA LETTER SMALL E
3267	<E6>	/d032/d032/d047/d168	KATAKANA LETTER E
3268	<o6>	/d032/d032/d047/d169	KATAKANA LETTER SMALL O
3269	<O6>	/d032/d032/d047/d170	KATAKANA LETTER O
3270	<Ka>	/d032/d032/d047/d171	KATAKANA LETTER KA
3271	<Ga>	/d032/d032/d047/d172	KATAKANA LETTER GA
3272	<Ki>	/d032/d032/d047/d173	KATAKANA LETTER KI
3273	<Gi>	/d032/d032/d047/d174	KATAKANA LETTER GI
3274	<Ku>	/d032/d032/d047/d175	KATAKANA LETTER KU
3275	<Gu>	/d032/d032/d047/d176	KATAKANA LETTER GU
3276	<Ke>	/d032/d032/d047/d177	KATAKANA LETTER KE
3277	<Ge>	/d032/d032/d047/d178	KATAKANA LETTER GE
3278	<Ko>	/d032/d032/d047/d179	KATAKANA LETTER KO
3279	<Go>	/d032/d032/d047/d180	KATAKANA LETTER GO
3280	<Sa>	/d032/d032/d047/d181	KATAKANA LETTER SA
3281	<Za>	/d032/d032/d047/d182	KATAKANA LETTER ZA
3282	<Si>	/d032/d032/d047/d183	KATAKANA LETTER SI
3283	<Zi>	/d032/d032/d047/d184	KATAKANA LETTER ZI
3284	<Su>	/d032/d032/d047/d185	KATAKANA LETTER SU
3285	<Zu>	/d032/d032/d047/d186	KATAKANA LETTER ZU
3286	<Se>	/d032/d032/d047/d187	KATAKANA LETTER SE
3287	<Ze>	/d032/d032/d047/d188	KATAKANA LETTER ZE
3288	<So>	/d032/d032/d047/d189	KATAKANA LETTER SO
3289	<Zo>	/d032/d032/d047/d190	KATAKANA LETTER ZO
3290	<Ta>	/d032/d032/d047/d191	KATAKANA LETTER TA
3291	<Da>	/d032/d032/d047/d192	KATAKANA LETTER DA
3292	<Ti>	/d032/d032/d047/d193	KATAKANA LETTER TI
3293	<Di>	/d032/d032/d047/d194	KATAKANA LETTER DI
3294	<TU>	/d032/d032/d047/d195	KATAKANA LETTER SMALL TU
3295	<Tu>	/d032/d032/d047/d196	KATAKANA LETTER TU
3296	<Du>	/d032/d032/d047/d197	KATAKANA LETTER DU
3297	<Te>	/d032/d032/d047/d198	KATAKANA LETTER TE
3298	<De>	/d032/d032/d047/d199	KATAKANA LETTER DE
3299	<To>	/d032/d032/d047/d200	KATAKANA LETTER TO
3300	<Do>	/d032/d032/d047/d201	KATAKANA LETTER DO
3301	<Na>	/d032/d032/d047/d202	KATAKANA LETTER NA
3302	<Ni>	/d032/d032/d047/d203	KATAKANA LETTER NI
3303	<Nu>	/d032/d032/d047/d204	KATAKANA LETTER NU
3304	<Ne>	/d032/d032/d047/d205	KATAKANA LETTER NE
3305	<No>	/d032/d032/d047/d206	KATAKANA LETTER NO
3306	<Ha>	/d032/d032/d047/d207	KATAKANA LETTER HA
3307	<Ba>	/d032/d032/d047/d208	KATAKANA LETTER BA
3308	<Pa>	/d032/d032/d047/d209	KATAKANA LETTER PA
3309	<Hi>	/d032/d032/d047/d210	KATAKANA LETTER HI
3310	<Bi>	/d032/d032/d047/d211	KATAKANA LETTER BI
3311	<Pi>	/d032/d032/d047/d212	KATAKANA LETTER PI
3312	<Hu>	/d032/d032/d047/d213	KATAKANA LETTER HU
3313	<Bu>	/d032/d032/d047/d214	KATAKANA LETTER BU
3314	<Pu>	/d032/d032/d047/d215	KATAKANA LETTER PU
3315	<He>	/d032/d032/d047/d216	KATAKANA LETTER HE

1
1

3316	<Be>	/d032/d032/d047/d217	KATAKANA LETTER BE	
3317	<Pe>	/d032/d032/d047/d218	KATAKANA LETTER PE	
3318	<Ho>	/d032/d032/d047/d219	KATAKANA LETTER HO	
3319	<Bo>	/d032/d032/d047/d220	KATAKANA LETTER BO	
3320	<Po>	/d032/d032/d047/d221	KATAKANA LETTER PO	
3321	<Ma>	/d032/d032/d047/d222	KATAKANA LETTER MA	
3322	<Mi>	/d032/d032/d047/d223	KATAKANA LETTER MI	
3323	<Mu>	/d032/d032/d047/d224	KATAKANA LETTER MU	
3324	<Me>	/d032/d032/d047/d225	KATAKANA LETTER ME	
3325	<Mo>	/d032/d032/d047/d226	KATAKANA LETTER MO	
3326	<YA>	/d032/d032/d047/d227	KATAKANA LETTER SMALL YA	
3327	<Ya>	/d032/d032/d047/d228	KATAKANA LETTER YA	
3328	<YU>	/d032/d032/d047/d229	KATAKANA LETTER SMALL YU	
3329	<Yu>	/d032/d032/d047/d230	KATAKANA LETTER YU	
3330	<YO>	/d032/d032/d047/d231	KATAKANA LETTER SMALL YO	
3331	<Yo>	/d032/d032/d047/d232	KATAKANA LETTER YO	
3332	<Ra>	/d032/d032/d047/d233	KATAKANA LETTER RA	
3333	<Ri>	/d032/d032/d047/d234	KATAKANA LETTER RI	
3334	<Ru>	/d032/d032/d047/d235	KATAKANA LETTER RU	
3335	<Re>	/d032/d032/d047/d236	KATAKANA LETTER RE	
3336	<Ro>	/d032/d032/d047/d237	KATAKANA LETTER RO	
3337	<WA>	/d032/d032/d047/d238	KATAKANA LETTER SMALL WA	
3338	<Wa>	/d032/d032/d047/d239	KATAKANA LETTER WA	
3339	<Wi>	/d032/d032/d047/d240	KATAKANA LETTER WI	
3340	<We>	/d032/d032/d047/d241	KATAKANA LETTER WE	
3341	<Wo>	/d032/d032/d047/d242	KATAKANA LETTER WO	
3342	<N6>	/d032/d032/d047/d243	KATAKANA LETTER N	
3343	<Vu>	/d032/d032/d047/d244	KATAKANA LETTER VU	
3344	<KA>	/d032/d032/d047/d245	KATAKANA LETTER SMALL KA	
3345	<KE>	/d032/d032/d047/d246	KATAKANA LETTER SMALL KE	
3346	<-6>	/d032/d032/d047/d252	HIRAGANA-KATAKANA PROLONGED SOUND MARK	
3347	<*6>	/d032/d032/d047/d253	KATAKANA ITERATION MARK	
3348	<+6>	/d032/d032/d047/d254	KATAKANA VOICED ITERATION MARK	
3349	<ff>	/d032/d032/d060/d040	LATIN SMALL LIGATURE FF	
3350	<fi>	/d032/d032/d060/d041	LATIN SMALL LIGATURE FI	
3351	<fl>	/d032/d032/d060/d042	LATIN SMALL LIGATURE FL	
3352	<ft>	/d032/d032/d060/d045	LATIN SMALL LIGATURE FT	
3353	<st>	/d032/d032/d060/d046	LATIN SMALL LIGATURE ST	
3354	<Iu>	/d032/d032/d060/d048	INTEGRAL SIGN UPPER PART	
3355	<Il>	/d032/d032/d060/d049	INTEGRAL SIGN LOWER PART	
3356	<NU>	/d000/d128/d128/d128	NULL (NUL)	1
3357	<SH>	/d001/d128/d128/d128	START OF HEADING (SOH)	1
3358	<SX>	/d002/d128/d128/d128	START OF TEXT (STX)	1
3359	<EX>	/d003/d128/d128/d128	END OF TEXT (ETX)	1
3360	<ET>	/d004/d128/d128/d128	END OF TRANSMISSION (EOT)	1
3361	<EQ>	/d005/d128/d128/d128	ENQUIRY (ENQ)	1
3362	<AK>	/d006/d128/d128/d128	ACKNOWLEDGE (ACK)	1
3363	<BL>	/d007/d128/d128/d128	BELL (BEL)	1
3364	<BS>	/d008/d128/d128/d128	BACKSPACE (BS)	1
3365	<HT>	/d009/d128/d128/d128	CHARACTER TABULATION (HT)	1
3366	<LF>	/d010/d128/d128/d128	LINE FEED (LF)	1
3367	<VT>	/d011/d128/d128/d128	LINE TABULATION (VT)	1
3368	<FF>	/d012/d128/d128/d128	FORM FEED (FF)	1
3369	<CR>	/d013/d128/d128/d128	CARRIAGE RETURN (CR)	1
3370	<SO>	/d014/d128/d128/d128	SHIFT OUT (SO)	1
3371	<SI>	/d015/d128/d128/d128	SHIFT IN (SI)	1
3372	<DL>	/d016/d128/d128/d128	DATALINK ESCAPE (DLE)	1

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

3373	<D1>	/d017/d128/d128/d128	DEVICE CONTROL ONE (DC1)	1
3374	<D2>	/d018/d128/d128/d128	DEVICE CONTROL TWO (DC2)	1
3375	<D3>	/d019/d128/d128/d128	DEVICE CONTROL THREE (DC3)	1
3376	<D4>	/d020/d128/d128/d128	DEVICE CONTROL FOUR (DC4)	1
3377	<NK>	/d021/d128/d128/d128	NEGATIVE ACKNOWLEDGE (NAK)	1
3378	<SY>	/d022/d128/d128/d128	SYNCHRONOUS IDLE (SYN)	1
3379	<EB>	/d023/d128/d128/d128	END OF TRANSMISSION BLOCK (ETB)	1
3380	<CN>	/d024/d128/d128/d128	CANCEL (CAN)	1
3381		/d025/d128/d128/d128	END OF MEDIUM (EM)	1
3382	<SB>	/d026/d128/d128/d128	SUBSTITUTE (SUB)	1
3383	<EC>	/d027/d128/d128/d128	ESCAPE (ESC)	1
3384	<FS>	/d028/d128/d128/d128	FILE SEPARATOR (IS4)	1
3385	<GS>	/d029/d128/d128/d128	GROUP SEPARATOR (IS3)	1
3386	<RS>	/d030/d128/d128/d128	RECORD SEPARATOR (IS2)	1
3387	<US>	/d031/d128/d128/d128	UNIT SEPARATOR (IS1)	1
3388	<DT>	/d127/d128/d128/d128	DELETE (DEL)	1
3389	<PA>	/d128/d128/d128/d128	PADDING CHARACTER (PAD)	1
3390	<HO>	/d129/d128/d128/d128	HIGH OCTET PRESET (HOP)	1
3391	<BH>	/d130/d128/d128/d128	BREAK PERMITTED HERE (BPH)	1
3392	<NH>	/d131/d128/d128/d128	NO BREAK HERE (NBH)	1
3393	<IN>	/d132/d128/d128/d128	INDEX (IND)	1
3394	<NL>	/d133/d128/d128/d128	NEXT LINE (NEL)	1
3395	<SA>	/d134/d128/d128/d128	START OF SELECTED AREA (SSA)	1
3396	<ES>	/d135/d128/d128/d128	END OF SELECTED AREA (ESA)	1
3397	<HS>	/d136/d128/d128/d128	CHARACTER TABULATION SET (HTS)	1
3398	<HJ>	/d137/d128/d128/d128	CHARACTER TABULATION WITH JUSTIFICATION (HTJ)	1
3399	<VS>	/d138/d128/d128/d128	LINE TABULATION SET (VTS)	1
3400	<PD>	/d139/d128/d128/d128	PARTIAL LINE FORWARD (PLD)	1
3401	<PU>	/d140/d128/d128/d128	PARTIAL LINE BACKWARD (PLU)	1
3402	<RI>	/d141/d128/d128/d128	REVERSE LINE FEED (RI)	1
3403	<S2>	/d142/d128/d128/d128	SINGLE-SHIFT TWO (SS2)	1
3404	<S3>	/d143/d128/d128/d128	SINGLE-SHIFT THREE (SS3)	1
3405	<DC>	/d144/d128/d128/d128	DEVICE CONTROL STRING (DCS)	1
3406	<P1>	/d145/d128/d128/d128	PRIVATE USE ONE (PU1)	1
3407	<P2>	/d146/d128/d128/d128	PRIVATE USE TWO (PU2)	1
3408	<TS>	/d147/d128/d128/d128	SET TRANSMIT STATE (STS)	1
3409	<CC>	/d148/d128/d128/d128	CANCEL CHARACTER (CCH)	1
3410	<MW>	/d149/d128/d128/d128	MESSAGE WAITING (MW)	1
3411	<SG>	/d150/d128/d128/d128	START OF GUARDED AREA (SPA)	1
3412	<EG>	/d151/d128/d128/d128	END OF GUARDED AREA (EPA)	1
3413	<SS>	/d152/d128/d128/d128	START OF STRING (SOS)	1
3414	<GC>	/d153/d128/d128/d128	SINGLE GRAPHIC CHARACTER INTRODUCER (SGCI)	1
3415	<SC>	/d154/d128/d128/d128	SINGLE CHARACTER INTRODUCER (SCI)	1
3416	<CI>	/d155/d128/d128/d128	CONTROL SEQUENCE INTRODUCER (CSI)	1
3417	<ST>	/d156/d128/d128/d128	STRING TERMINATOR (ST)	1
3418	<OC>	/d157/d128/d128/d128	OPERATING SYSTEM COMMAND (OSC)	1
3419	<PM>	/d158/d128/d128/d128	PRIVACY MESSAGE (PM)	1
3420	<AC>	/d159/d128/d128/d128	APPLICATION PROGRAM COMMAND (APC)	1
3421	<_>	/d032/d032/d052/d032	indicates unfinished	
3422	<"!>	/d032/d032/d052/d033	NON-SPACING GRAVE ACCENT (ISO IR 70 193)	
3423	<"'>	/d032/d032/d052/d034	NON-SPACING ACUTE ACCENT (ISO IR 70 194)	
3424	<"/>>	/d032/d032/d052/d035	NON-SPACING CIRCUMFLEX ACCENT (ISO IR 70 195)	
3425	<"?>	/d032/d032/d052/d036	NON-SPACING TILDE (ISO IR 70 196)	
3426	<"->	/d032/d032/d052/d037	NON-SPACING MACRON (ISO IR 70 197)	
3427	<"(>	/d032/d032/d052/d038	NON-SPACING BREVE (ISO IR 70 198)	
3428	<".>	/d032/d032/d052/d039	NON-SPACING DOT ABOVE (ISO IR 70 199)	
3429	<":>	/d032/d032/d052/d040	NON-SPACING DIAERESIS (ISO IR 70 200)	

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

3430	<"//>	/d032/d032/d052/d041	NON-SPACING SOLIDUS (ISO IR 99 201)
3431	<"0>	/d032/d032/d052/d042	NON-SPACING RING ABOVE (ISO IR 70 202)
3432	<" ,>	/d032/d032/d052/d043	NON-SPACING CEDILLA (ISO IR 70 203)
3433	<"_>	/d032/d032/d052/d044	NON-SPACING UNDERLINE (ISO IR 99 216)
3434	<" ">	/d032/d032/d052/d045	NON-SPACING DOUBLE ACCUTE ACCENT (ISO IR 70 205)
3435	<"<>	/d032/d032/d052/d046	NON-SPACING CARON (ISO IR 70 207)
3436	<" ;>	/d032/d032/d052/d047	NON-SPACING OGONEK (ISO IR 53 208)
3437	<"=>	/d032/d032/d052/d048	NON-SPACING DOUBLE UNDERLINE (ISO IR 53 217)
3438	<"1>	/d032/d032/d052/d049	NON-SPACING DIAERESIS WITH ACCENT (ISO IR 70 192)
3439	#		
3440	<"2>	/d032/d032/d052/d050	NON-SPACING UMLAUT (ISO 5426 201)
3441	<Fd>	/d032/d032/d052/d051	FILLED FORWARD DIAGONAL
3442	#		(ANSI X3.110-1983 218)
3443	<Bd>	/d032/d032/d052/d052	FILLED BACKWARD DIAGONAL
3444	#		(ANSI X3.110-1983 219)
3445	<Fl>	/d032/d032/d052/d053	Dutch guilder sign (IBM CP 437 159)
3446		/d032/d032/d052/d054	Italian Lira sign (HP ROMAN 8 175)
3447	< //f>	/d032/d032/d052/d055	VULGAR FRACTION BAR (MacIntosh 218)
3448	<0s>	/d032/d032/d052/d056	SUBSCRIPT ZERO (ISO IR 50 096)
3449	<1s>	/d032/d032/d052/d057	SUBSCRIPT ONE (ISO IR 50 097)
3450	<2s>	/d032/d032/d052/d058	SUBSCRIPT TWO (ISO IR 50 098)
3451	<3s>	/d032/d032/d052/d059	SUBSCRIPT THREE (ISO IR 50 099)
3452	<4s>	/d032/d032/d052/d060	SUBSCRIPT FOUR (ISO IR 50 100)
3453	<5s>	/d032/d032/d052/d061	SUBSCRIPT FIVE (ISO IR 50 101)
3454	<6s>	/d032/d032/d052/d062	SUBSCRIPT SIX (ISO IR 50 102)
3455	<7s>	/d032/d032/d052/d063	SUBSCRIPT SEVEN (ISO IR 50 103)
3456	<8s>	/d032/d032/d052/d064	SUBSCRIPT EIGHT (ISO IR 50 104)
3457	<9s>	/d032/d032/d052/d065	SUBSCRIPT NINE (ISO IR 50 105)
3458	<0S>	/d032/d032/d052/d066	SUPERSCRIPIT ZERO (ISO IR 50 112)
3459	<4S>	/d032/d032/d052/d067	SUPERSCRIPIT FOUR (ISO IR 50 116)
3460	<5S>	/d032/d032/d052/d068	SUPERSCRIPIT FIVE (ISO IR 50 117)
3461	<6S>	/d032/d032/d052/d069	SUPERSCRIPIT SIX (ISO IR 50 118)
3462	<7S>	/d032/d032/d052/d070	SUPERSCRIPIT SEVEN (ISO IR 50 119)
3463	<8S>	/d032/d032/d052/d071	SUPERSCRIPIT EIGHT (ISO IR 50 120)
3464	<9S>	/d032/d032/d052/d072	SUPERSCRIPIT NINE (ISO IR 50 121)
3465	<+S>	/d032/d032/d052/d073	SUPERSCRIPIT PLUS (ISO IR 50 106)
3466	<-S>	/d032/d032/d052/d074	SUPERSCRIPIT MINUS (ISO IR 50 107)
3467	<1h>	/d032/d032/d052/d075	ABSTRACT SYMBOL H ONE (HOOK)
3468	#		(JIS C 6229-1984 060)
3469	<2h>	/d032/d032/d052/d076	ABSTRACT SYMBOL H TWO (FORK)
3470	#		(JIS C 6229-1984 093)
3471	<3h>	/d032/d032/d052/d077	ABSTRACT SYMBOL H THREE (CHAIR)
3472	#		(JIS C 6229-1984 062)
3473	<4h>	/d032/d032/d052/d078	ABSTRACT SYMBOL H FOUR (LONG VERTICAL MARK)
3474	#		(JIS C 6229-1984 125)
3475	<1j>	/d032/d032/d052/d079	SYMBOL ONE (ISO 2033-1983 058)
3476	<2j>	/d032/d032/d052/d080	SYMBOL TWO (ISO 2033-1983 059)
3477	<3j>	/d032/d032/d052/d081	SYMBOL THREE (ISO 2033-1983 060)
3478	<4j>	/d032/d032/d052/d082	SYMBOL FOUR (ISO 2033-1983 061)
3479	<UA>	/d032/d032/d052/d083	Unit space A (ISO IR 8-1 064)
3480	<UB>	/d032/d032/d052/d084	Unit space B (ISO IR 8-1 096)
3481	<yf>	/d032/d032/d052/d085	ARABIC LETTER YEH FINAL (CODAR U 090)
3482	<yr>	/d032/d032/d052/d086	OLD NORSE YR (DIN 31624 251)
3483	<.6>	/d032/d032/d052/d087	KATAKANA FULL STOP (JIS C 6220 033)
3484	<<6>	/d032/d032/d052/d088	KATAKANA OPENING BRACKET (JIS C 6220 034)
3485	</>6>	/d032/d032/d052/d089	KATAKANA CLOSING BRACKET (JIS C 6220 035)
3486	< ,6>	/d032/d032/d052/d090	KATAKANA COMMA (JIS C 6220 036)

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.


```

3487 <&6>      /d032/d032/d052/d091  KATAKANA CONJUNCTION SYMBOL (JIS C 6220 037)
3488 <(S>      /d032/d032/d052/d092  LEFT PARENTHESIS SUPERScript
3489 #                (CSA Z243.4-1985-gr 168)
3490 <)S>      /d032/d032/d052/d093  RIGHT PARENTHESIS SUPERScript
3491 #                (CSA Z243.4-1985-gr 169)
3492 END CHARMAP

```

3493 **F.5.2 iso_8859-1 Charmap**

```

3494 <escape_char> /
3495 <mb_cur_max> 1
3496 CHARMAP
3497 <NUL>          /d000 NULL (NUL)
3498 <SOH>          /d001 START OF HEADING (SOH)
3499 <STX>          /d002 START OF TEXT (STX)
3500 <ETX>          /d003 END OF TEXT (ETX)
3501 <EOT>          /d004 END OF TRANSMISSION (EOT)
3502 <ENQ>          /d005 ENQUIRY (ENQ)
3503 <ACK>          /d006 ACKNOWLEDGE (ACK)
3504 <alert>        /d007 BELL (BEL)
3505 <BEL>          /d007 BELL (BEL)
3506 <backspace>    /d008 BACKSPACE (BS)
3507 <tab>          /d009 CHARACTER TABULATION (HT)
3508 <newline>      /d010 LINE FEED (LF)
3509 <vertical-tab> /d011 LINE TABULATION (VT)
3510 <form-feed>    /d012 FORM FEED (FF)
3511 <carriage-return> /d013 CARRIAGE RETURN (CR)
3512 <DLE>          /d016 DATALINK ESCAPE (DLE)
3513 <DC1>          /d017 DEVICE CONTROL ONE (DC1)
3514 <DC2>          /d018 DEVICE CONTROL TWO (DC2)
3515 <DC3>          /d019 DEVICE CONTROL THREE (DC3)
3516 <DC4>          /d020 DEVICE CONTROL FOUR (DC4)
3517 <NAK>          /d021 NEGATIVE ACKNOWLEDGE (NAK)
3518 <SYN>          /d022 SYNCHRONOUS IDLE (SYN)
3519 <ETB>          /d023 END OF TRANSMISSION BLOCK (ETB)
3520 <CAN>          /d024 CANCEL (CAN)
3521 <SUB>          /d026 SUBSTITUTE (SUB)
3522 <ESC>          /d027 ESCAPE (ESC)
3523 <IS4>          /d028 FILE SEPARATOR (IS4)
3524 <IS3>          /d029 GROUP SEPARATOR (IS3)
3525 <intro>        /d029 GROUP SEPARATOR (IS3)
3526 <IS2>          /d030 RECORD SEPARATOR (IS2)
3527 <IS1>          /d031 UNIT SEPARATOR (IS1)
3528 <DEL>          /dl27 DELETE (DEL)
3529 <space>        /d032 SPACE
3530 <exclamation-mark> /d033 EXCLAMATION MARK
3531 <quotation-mark> /d034 QUOTATION MARK
3532 <number-sign>    /d035 NUMBER SIGN
3533 <dollar-sign>    /d036 DOLLAR SIGN
3534 <percent-sign>   /d037 PERCENT SIGN
3535 <ampersand>       /d038 AMPERSAND
3536 <apostrophe>    /d039 APOSTROPHE
3537 <left-parenthesis> /d040 LEFT PARENTHESIS
3538 <right-parenthesis> /d041 RIGHT PARENTHESIS
3539 <asterisk>       /d042 ASTERISK
3540 <plus-sign>      /d043 PLUS SIGN

```

1

1

3541	<comma>	/d044	COMMA
3542	<hyphen>	/d045	HYPHEN-MINUS
3543	<hyphen-minus>	/d045	HYPHEN-MINUS
3544	<period>	/d046	FULL STOP
3545	<full-stop>	/d046	FULL STOP
3546	<slash>	/d047	SOLIDUS
3547	<solidus>	/d047	SOLIDUS
3548	<zero>	/d048	DIGIT ZERO
3549	<one>	/d049	DIGIT ONE
3550	<two>	/d050	DIGIT TWO
3551	<three>	/d051	DIGIT THREE
3552	<four>	/d052	DIGIT FOUR
3553	<five>	/d053	DIGIT FIVE
3554	<six>	/d054	DIGIT SIX
3555	<seven>	/d055	DIGIT SEVEN
3556	<eight>	/d056	DIGIT EIGHT
3557	<nine>	/d057	DIGIT NINE
3558	<colon>	/d058	COLON
3559	<semicolon>	/d059	SEMICOLON
3560	<less-than-sign>	/d060	LESS-THAN SIGN
3561	<equals-sign>	/d061	EQUALS SIGN
3562	<greater-than-sign>	/d062	GREATER-THAN SIGN
3563	<question-mark>	/d063	QUESTION MARK
3564	<commercial-at>	/d064	COMMERCIAL AT
3565	<left-square-bracket>	/d091	LEFT SQUARE BRACKET
3566	<reverse-solidus>	/d092	REVERSE SOLIDUS
3567	<backslash>	/d092	REVERSE SOLIDUS
3568	<right-square-bracket>	/d093	RIGHT SQUARE BRACKET
3569	<circumflex-accent>	/d094	CIRCUMFLEX ACCENT
3570	<low-line>	/d095	LOW LINE
3571	<underscore>	/d095	LOW LINE
3572	<grave-accent>	/d096	GRAVE ACCENT
3573	<left-curly-bracket>	/d123	LEFT CURLY BRACKET
3574	<vertical-line>	/d124	VERTICAL LINE
3575	<right-curly-bracket>	/d125	RIGHT CURLY BRACKET
3576	<tilde>	/d126	TILDE
3577	<SP>	/d032	SPACE
3578	<!>	/d033	EXCLAMATION MARK
3579	<">	/d034	QUOTATION MARK
3580	<Nb>	/d035	NUMBER SIGN
3581	<DO>	/d036	DOLLAR SIGN
3582	<%>	/d037	PERCENT SIGN
3583	<&>	/d038	AMPERSAND
3584	<'>	/d039	APOSTROPHE
3585	<(>	/d040	LEFT PARENTHESIS
3586	<)>	/d041	RIGHT PARENTHESIS
3587	<*>	/d042	ASTERISK
3588	<+>	/d043	PLUS SIGN
3589	<,>	/d044	COMMA
3590	<->	/d045	HYPHEN-MINUS
3591	<.>	/d046	FULL STOP
3592	<//>	/d047	SOLIDUS
3593	<0>	/d048	DIGIT ZERO
3594	<1>	/d049	DIGIT ONE
3595	<2>	/d050	DIGIT TWO
3596	<3>	/d051	DIGIT THREE
3597	<4>	/d052	DIGIT FOUR

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

3598	<5>	/d053	DIGIT FIVE
3599	<6>	/d054	DIGIT SIX
3600	<7>	/d055	DIGIT SEVEN
3601	<8>	/d056	DIGIT EIGHT
3602	<9>	/d057	DIGIT NINE
3603	<:>	/d058	COLON
3604	<;>	/d059	SEMICOLON
3605	<<>	/d060	LESS-THAN SIGN
3606	<=>	/d061	EQUALS SIGN
3607	</>>	/d062	GREATER-THAN SIGN
3608	<?>	/d063	QUESTION MARK
3609	<At>	/d064	COMMERCIAL AT
3610	<A>	/d065	LATIN CAPITAL LETTER A
3611		/d066	LATIN CAPITAL LETTER B
3612	<C>	/d067	LATIN CAPITAL LETTER C
3613	<D>	/d068	LATIN CAPITAL LETTER D
3614	<E>	/d069	LATIN CAPITAL LETTER E
3615	<F>	/d070	LATIN CAPITAL LETTER F
3616	<G>	/d071	LATIN CAPITAL LETTER G
3617	<H>	/d072	LATIN CAPITAL LETTER H
3618	<I>	/d073	LATIN CAPITAL LETTER I
3619	<J>	/d074	LATIN CAPITAL LETTER J
3620	<K>	/d075	LATIN CAPITAL LETTER K
3621	<L>	/d076	LATIN CAPITAL LETTER L
3622	<M>	/d077	LATIN CAPITAL LETTER M
3623	<N>	/d078	LATIN CAPITAL LETTER N
3624	<O>	/d079	LATIN CAPITAL LETTER O
3625	<P>	/d080	LATIN CAPITAL LETTER P
3626	<Q>	/d081	LATIN CAPITAL LETTER Q
3627	<R>	/d082	LATIN CAPITAL LETTER R
3628	<S>	/d083	LATIN CAPITAL LETTER S
3629	<T>	/d084	LATIN CAPITAL LETTER T
3630	<U>	/d085	LATIN CAPITAL LETTER U
3631	<V>	/d086	LATIN CAPITAL LETTER V
3632	<W>	/d087	LATIN CAPITAL LETTER W
3633	<X>	/d088	LATIN CAPITAL LETTER X
3634	<Y>	/d089	LATIN CAPITAL LETTER Y
3635	<Z>	/d090	LATIN CAPITAL LETTER Z
3636	<<(>	/d091	LEFT SQUARE BRACKET
3637	</////>	/d092	REVERSE SOLIDUS
3638	<)/>>	/d093	RIGHT SQUARE BRACKET
3639	<' />>	/d094	CIRCUMFLEX ACCENT
3640	<_>	/d095	LOW LINE
3641	<' !>	/d096	GRAVE ACCENT
3642	<a>	/d097	LATIN SMALL LETTER A
3643		/d098	LATIN SMALL LETTER B
3644	<c>	/d099	LATIN SMALL LETTER C
3645	<d>	/d100	LATIN SMALL LETTER D
3646	<e>	/d101	LATIN SMALL LETTER E
3647	<f>	/d102	LATIN SMALL LETTER F
3648	<g>	/d103	LATIN SMALL LETTER G
3649	<h>	/d104	LATIN SMALL LETTER H
3650	<i>	/d105	LATIN SMALL LETTER I
3651	<j>	/d106	LATIN SMALL LETTER J
3652	<k>	/d107	LATIN SMALL LETTER K
3653	<l>	/d108	LATIN SMALL LETTER L
3654	<m>	/d109	LATIN SMALL LETTER M

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

3655	<n>	/d110	LATIN SMALL LETTER N
3656	<o>	/d111	LATIN SMALL LETTER O
3657	<p>	/d112	LATIN SMALL LETTER P
3658	<q>	/d113	LATIN SMALL LETTER Q
3659	<r>	/d114	LATIN SMALL LETTER R
3660	<s>	/d115	LATIN SMALL LETTER S
3661	<t>	/d116	LATIN SMALL LETTER T
3662	<u>	/d117	LATIN SMALL LETTER U
3663	<v>	/d118	LATIN SMALL LETTER V
3664	<w>	/d119	LATIN SMALL LETTER W
3665	<x>	/d120	LATIN SMALL LETTER X
3666	<y>	/d121	LATIN SMALL LETTER Y
3667	<z>	/d122	LATIN SMALL LETTER Z
3668	<(!>	/d123	LEFT CURLY BRACKET
3669	<!!>	/d124	VERTICAL LINE
3670	<!)>	/d125	RIGHT CURLY BRACKET
3671	<'>	/d126	TILDE
3672	<NS>	/d160	NO-BREAK SPACE
3673	<!I>	/d161	INVERTED EXCLAMATION MARK
3674	<Ct>	/d162	CENT SIGN
3675	<Pd>	/d163	POUND SIGN
3676	<Cu>	/d164	CURRENCY SIGN
3677	<Ye>	/d165	YEN SIGN
3678	<BB>	/d166	BROKEN BAR
3679	<SE>	/d167	SECTION SIGN
3680	<' :>	/d168	DIAERESIS
3681	<Co>	/d169	COPYRIGHT SIGN
3682	<-a>	/d170	FEMININE ORDINAL INDICATOR
3683	<<<>	/d171	LEFT POINTING DOUBLE ANGLE QUOTATION MARK
3684	<NO>	/d172	NOT SIGN
3685	<-->	/d173	SOFT HYPHEN
3686	<Rg>	/d174	REGISTERED SIGN
3687	<' ->	/d175	MACRON
3688	<DG>	/d176	DEGREE SIGN
3689	<+->	/d177	PLUS-MINUS SIGN
3690	<2S>	/d178	SUPERSCRIFT TWO
3691	<3S>	/d179	SUPERSCRIFT THREE
3692	<' ' >	/d180	ACUTE ACCENT
3693	<My>	/d181	MICRO SIGN
3694	<PI>	/d182	PILCROW SIGN
3695	<.M>	/d183	MIDDLE DOT
3696	<' ,>	/d184	CEDILLA
3697	<1S>	/d185	SUPERSCRIFT ONE
3698	<-o>	/d186	MASCULINE ORDINAL INDICATOR
3699	</>>>>	/d187	RIGHT POINTING DOUBLE ANGLE QUOTATION MARK
3700	<14>	/d188	VULGAR FRACTION ONE QUARTER
3701	<12>	/d189	VULGAR FRACTION ONE HALF
3702	<34>	/d190	VULGAR FRACTION THREE QUARTERS
3703	<?I>	/d191	INVERTED QUESTION MARK
3704	<A!>	/d192	LATIN CAPITAL LETTER A WITH GRAVE
3705	<A'>	/d193	LATIN CAPITAL LETTER A WITH ACUTE
3706	<A/>>	/d194	LATIN CAPITAL LETTER A WITH CIRCUMFLEX
3707	<A?>	/d195	LATIN CAPITAL LETTER A WITH TILDE
3708	<A:>	/d196	LATIN CAPITAL LETTER A WITH DIAERESIS
3709	<AA>	/d197	LATIN CAPITAL LETTER A WITH RING ABOVE
3710	<AE>	/d198	LATIN CAPITAL LETTER AE
3711	<C,>	/d199	LATIN CAPITAL LETTER C WITH CEDILLA

1

Copyright © 1991 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

3712	<E!>	/d200	LATIN CAPITAL LETTER E WITH GRAVE
3713	<E'>	/d201	LATIN CAPITAL LETTER E WITH ACUTE
3714	<E/>>	/d202	LATIN CAPITAL LETTER E WITH CIRCUMFLEX
3715	<E:>	/d203	LATIN CAPITAL LETTER E WITH DIAERESIS
3716	<I!>	/d204	LATIN CAPITAL LETTER I WITH GRAVE
3717	<I'>	/d205	LATIN CAPITAL LETTER I WITH ACUTE
3718	<I/>>	/d206	LATIN CAPITAL LETTER I WITH CIRCUMFLEX
3719	<I:>	/d207	LATIN CAPITAL LETTER I WITH DIAERESIS
3720	<D->	/d208	LATIN CAPITAL LETTER ETH (Icelandic)
3721	<N?>	/d209	LATIN CAPITAL LETTER N WITH TILDE
3722	<O!>	/d210	LATIN CAPITAL LETTER O WITH GRAVE
3723	<O'>	/d211	LATIN CAPITAL LETTER O WITH ACUTE
3724	<O/>>	/d212	LATIN CAPITAL LETTER O WITH CIRCUMFLEX
3725	<O?>	/d213	LATIN CAPITAL LETTER O WITH TILDE
3726	<O:>	/d214	LATIN CAPITAL LETTER O WITH DIAERESIS
3727	<*X>	/d215	MULTIPLICATION SIGN
3728	<O//>	/d216	LATIN CAPITAL LETTER O WITH STROKE
3729	<U!>	/d217	LATIN CAPITAL LETTER U WITH GRAVE
3730	<U'>	/d218	LATIN CAPITAL LETTER U WITH ACUTE
3731	<U/>>	/d219	LATIN CAPITAL LETTER U WITH CIRCUMFLEX
3732	<U:>	/d220	LATIN CAPITAL LETTER U WITH DIAERESIS
3733	<Y'>	/d221	LATIN CAPITAL LETTER Y WITH ACUTE
3734	<TH>	/d222	LATIN CAPITAL LETTER THORN (Icelandic)
3735	<ss>	/d223	LATIN SMALL LETTER SHARP S (German)
3736	<a!>	/d224	LATIN SMALL LETTER A WITH GRAVE
3737	<a'>	/d225	LATIN SMALL LETTER A WITH ACUTE
3738	<a/>>	/d226	LATIN SMALL LETTER A WITH CIRCUMFLEX
3739	<a?>	/d227	LATIN SMALL LETTER A WITH TILDE
3740	<a:>	/d228	LATIN SMALL LETTER A WITH DIAERESIS
3741	<aa>	/d229	LATIN SMALL LETTER A WITH RING ABOVE
3742	<ae>	/d230	LATIN SMALL LETTER AE
3743	<c,>	/d231	LATIN SMALL LETTER C WITH CEDILLA
3744	<e!>	/d232	LATIN SMALL LETTER E WITH GRAVE
3745	<e'>	/d233	LATIN SMALL LETTER E WITH ACUTE
3746	<e/>>	/d234	LATIN SMALL LETTER E WITH CIRCUMFLEX
3747	<e:>	/d235	LATIN SMALL LETTER E WITH DIAERESIS
3748	<i!>	/d236	LATIN SMALL LETTER I WITH GRAVE
3749	<i'>	/d237	LATIN SMALL LETTER I WITH ACUTE
3750	<i/>>	/d238	LATIN SMALL LETTER I WITH CIRCUMFLEX
3751	<i:>	/d239	LATIN SMALL LETTER I WITH DIAERESIS
3752	<d->	/d240	LATIN SMALL LETTER ETH (Icelandic)
3753	<n?>	/d241	LATIN SMALL LETTER N WITH TILDE
3754	<o!>	/d242	LATIN SMALL LETTER O WITH GRAVE
3755	<o'>	/d243	LATIN SMALL LETTER O WITH ACUTE
3756	<o/>>	/d244	LATIN SMALL LETTER O WITH CIRCUMFLEX
3757	<o?>	/d245	LATIN SMALL LETTER O WITH TILDE
3758	<o:>	/d246	LATIN SMALL LETTER O WITH DIAERESIS
3759	<-:>	/d247	DIVISION SIGN
3760	<o//>	/d248	LATIN SMALL LETTER O WITH STROKE
3761	<u!>	/d249	LATIN SMALL LETTER U WITH GRAVE
3762	<u'>	/d250	LATIN SMALL LETTER U WITH ACUTE
3763	<u/>>	/d251	LATIN SMALL LETTER U WITH CIRCUMFLEX
3764	<u:>	/d252	LATIN SMALL LETTER U WITH DIAERESIS
3765	<y'>	/d253	LATIN SMALL LETTER Y WITH ACUTE
3766	<th>	/d254	LATIN SMALL LETTER THORN (Icelandic)
3767	<y:>	/d255	LATIN SMALL LETTER Y WITH DIAERESIS
3768	<NU>	/d000	NULL (NUL)

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

3769	<SH>	/d001	START OF HEADING (SOH)
3770	<SX>	/d002	START OF TEXT (STX)
3771	<EX>	/d003	END OF TEXT (ETX)
3772	<ET>	/d004	END OF TRANSMISSION (EOT)
3773	<EQ>	/d005	ENQUIRY (ENQ)
3774	<AK>	/d006	ACKNOWLEDGE (ACK)
3775	<BL>	/d007	BELL (BEL)
3776	<BS>	/d008	BACKSPACE (BS)
3777	<HT>	/d009	CHARACTER TABULATION (HT)
3778	<LF>	/d010	LINE FEED (LF)
3779	<VT>	/d011	LINE TABULATION (VT)
3780	<FF>	/d012	FORM FEED (FF)
3781	<CR>	/d013	CARRIAGE RETURN (CR)
3782	<SO>	/d014	SHIFT OUT (SO)
3783	<SI>	/d015	SHIFT IN (SI)
3784	<DL>	/d016	DATALINK ESCAPE (DLE)
3785	<D1>	/d017	DEVICE CONTROL ONE (DC1)
3786	<D2>	/d018	DEVICE CONTROL TWO (DC2)
3787	<D3>	/d019	DEVICE CONTROL THREE (DC3)
3788	<D4>	/d020	DEVICE CONTROL FOUR (DC4)
3789	<NK>	/d021	NEGATIVE ACKNOWLEDGE (NAK)
3790	<SY>	/d022	SYNCHRONOUS IDLE (SYN)
3791	<EB>	/d023	END OF TRANSMISSION BLOCK (ETB)
3792	<CN>	/d024	CANCEL (CAN)
3793		/d025	END OF MEDIUM (EM)
3794	<SB>	/d026	SUBSTITUTE (SUB)
3795	<EC>	/d027	ESCAPE (ESC)
3796	<FS>	/d028	FILE SEPARATOR (IS4)
3797	<GS>	/d029	GROUP SEPARATOR (IS3)
3798	<RS>	/d030	RECORD SEPARATOR (IS2)
3799	<US>	/d031	UNIT SEPARATOR (IS1)
3800	<DT>	/d127	DELETE (DEL)
3801	<PA>	/d128	PADDING CHARACTER (PAD)
3802	<HO>	/d129	HIGH OCTET PRESET (HOP)
3803	<BH>	/d130	BREAK PERMITTED HERE (BPH)
3804	<NH>	/d131	NO BREAK HERE (NBH)
3805	<IN>	/d132	INDEX (IND)
3806	<NL>	/d133	NEXT LINE (NEL)
3807	<SA>	/d134	START OF SELECTED AREA (SSA)
3808	<ES>	/d135	END OF SELECTED AREA (ESA)
3809	<HS>	/d136	CHARACTER TABULATION SET (HTS)
3810	<HJ>	/d137	CHARACTER TABULATION WITH JUSTIFICATION (HTJ)
3811	<VS>	/d138	LINE TABULATION SET (VTS)
3812	<PD>	/d139	PARTIAL LINE FORWARD (PLD)
3813	<PU>	/d140	PARTIAL LINE BACKWARD (PLU)
3814	<RI>	/d141	REVERSE LINE FEED (RI)
3815	<S2>	/d142	SINGLE-SHIFT TWO (SS2)
3816	<S3>	/d143	SINGLE-SHIFT THREE (SS3)
3817	<DC>	/d144	DEVICE CONTROL STRING (DCS)
3818	<P1>	/d145	PRIVATE USE ONE (PU1)
3819	<P2>	/d146	PRIVATE USE TWO (PU2)
3820	<TS>	/d147	SET TRANSMIT STATE (STS)
3821	<CC>	/d148	CANCEL CHARACTER (CCH)
3822	<MW>	/d149	MESSAGE WAITING (MW)
3823	<SG>	/d150	START OF GUARDED AREA (SPA)
3824	<EG>	/d151	END OF GUARDED AREA (EPA)
3825	<SS>	/d152	START OF STRING (SOS)

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

3826	<GC>	/d153	SINGLE GRAPHIC CHARACTER INTRODUCER (SGCI)
3827	<SC>	/d154	SINGLE CHARACTER INTRODUCER (SCI)
3828	<CI>	/d155	CONTROL SEQUENCE INTRODUCER (CSI)
3829	<ST>	/d156	STRING TERMINATOR (ST)
3830	<OC>	/d157	OPERATING SYSTEM COMMAND (OSC)
3831	<PM>	/d158	PRIVACY MESSAGE (PM)
3832	<AC>	/d159	APPLICATION PROGRAM COMMAND (APC)
3833	END CHARMAP		

Annex G (informative)

Balloting Instructions

This annex will not appear in the final standard. It is included in the draft to provide instructions for balloting that cannot be separated easily from the main document, as a cover letter might.

If you have received a copy of this draft before October 1991 it is important that you read this annex, whether you are an official member of the P1003.2 Balloting Group or not; comments on this draft are welcomed from all interested technical experts. **Your ballot is due to the IEEE office by 21 October 1991. This is not the date to postmark it—it is the date of receipt.**

Summary of Draft 11.2 Instructions

This is the fifth “recirculation draft” of P1003.2. The recirculation procedure is described in this annex. For this recirculation, we are accepting objections against any normative changes that occurred from Draft 11.1 to Draft 11.2 and the contents of the Unresolved Objections List, provided as a separate document from the draft.

This is the first ballot in which the draft is available for online review; see the Editor’s Notes for details on accessing this information.

Send your ballot and/or comments to:

IEEE Standards Office
Computer Society Secretariat
ATTN: P1003.2 Ballot (Anna Kaczmarek)
P.O. Box 1331
445 Hoes Lane
Piscataway, NJ 08855-1331

It would also be very helpful if you sent us your ballot in machine-readable form. Your official ballot must be returned via mail to the IEEE office; if we receive only the e-mail or diskette version, that version will not count as an official document. However, the online version would be a great help to ballot resolution. We can accept e-mail to the following address:

hlj@Posix.COM or uunet!posix!hlj

or IBM PC 3.5-inch/720K diskette (plain file) or Macintosh 3.5-inch diskette (plain text file [preferred], Word, or Write) or Sun-style QIC-24 cartridge tapes to:

Hal Jespersen, Chair P1003.2
 POSIX Software Group
 447 Lakeview Way
 Redwood City, CA 94062

Some degree of judgment is required in determining what actually changed in Draft 11.2. Use the diff marks as a guide, but they will frequently mark text that has no real normative changes. Please limit your objections to the actual changes: for example, if we change the `foo -x` option to `-y`, don't use that as an opportunity to object that we have no `-z` option. Your objection should only address why the `x` to `y` change is a problem. (We have been balloting for a long time now and it is time to tighten the consensus and finish this up.) If you find problems unrelated to changes, submit them as comments and they will be considered seriously in that category. Thanks for your cooperation on this.

Background on Balloting Procedures

The Balloting Group consists of over 160 technical experts who are members of the IEEE or the IEEE Computer Society; enrollment of individuals in this group has already been closed. There are also a few "parties of interest" who are not members of the IEEE or the Computer Society. Members of the Balloting Group are required to return ballots within the balloting period. Other individuals who may happen to read this draft are also encouraged to submit comments concerning this draft. The only real difference between members of the Balloting Group and other individuals submitting ballots is that *affirmative* ballots are only counted from Balloting Group members who are also IEEE or Computer Society members. (There are minimum requirements for the percentages of ballots returned and for affirmative ballots out of that group.) However, objections and nonbinding comments must be resolved if received from any individual, as follows:

- (1) Some objections or comments will result in changes to the standard. This will occur either by the publication of a list of changes or by the republication of an entire draft. The objections/comments are reviewed by a team from the P1003.2 working group, consisting of the Chair, Vice Chair, the Chair of the TCOS Standards Subcommittee, and one or more Technical Reviewers. The Technical Reviewers each have subject matter expertise in a particular area and are responsible for objection resolution in one or more sections.
- (2) Other objections/comments will not result in changes.
 - (a) Some are misunderstandings or cover portions of the document (front matter, informative annexes, rationale, editorial matters, etc.) that are not subject to balloting.
 - (b) Others are so vaguely worded that it is impossible to determine what changes would satisfy the objector. These are referred to as *Unresponsive*. (The Technical Reviewers will make a reasonable

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

effort to contact the objector to resolve this and get a newly worded objection.) Further examples of unresponsive submittals are those not marked as either *Objection* or *Comment*; those that do not identify the portion of the document that is being objected to (each objection must be separately labeled); those that object to material in a recirculation that has not changed and do not cite an unresolved objection; those that do not provide specific or general guidance on what changes would be required to resolve the objection.

- (c) Finally, others are valid technical points, but they would result in decreasing the consensus of the Balloting Group. (This judgment is made based on other ballots and on the experiences of the working group through almost five years of work and fifteen drafts preceding this one.) These are referred to as *Unresolved Objections*. Summaries of unresolved objections and their reasons for rejection are maintained throughout the balloting process, are circulated to members of the Balloting Group for their consideration, and are presented to the IEEE Standards Board when the final draft is offered for approval. Unresolved objections are only circulated to the balloting group when they are presented by members of the balloting group or by parties of interest. Unsolicited correspondence from outside these two groups may result in draft changes, but are not recirculated to the balloting group members.

Please ensure that you correctly characterize your ballot by providing one of the following:

- (1) Your IEEE member number
- (2) Your IEEE Computer Society affiliate number
- (3) If (1) or (2) don't apply, a statement that you are a "Party of Interest"

Ballot Resolution

The general procedure for resolving ballots is:

- (1) The balloting cuts off on 21 October 1991. This is a receipt date at the IEEE, not a postmark date. (Please do not telephone or FAX on 21 October 1991 and say that your specific comments will come later; late-arriving comments will not be considered as objections.) We will accept comments after that date, including direct e-mail to the working group officers or the Technical Reviewers, but they will be treated as comments only—not objections. And we don't guarantee a written response to these late submissions.
- (2) The ballots are put online and distributed to the Technical Reviewers.
- (3) If a ballot contains an objection, the balloter will be contacted individually by telephone, letter, or e-mail and the corrective action to be taken will be described (or negotiated). The personal contact will most likely not occur if the objection is very simple and obvious to fix or the balloter cannot be reached after a few reasonable attempts. Repeated failed

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

attempts to elicit a response from a balloter may result in an objection being considered unresponsive, based on the judgment of the working group chair. Once all objections in a ballot have been resolved, it becomes an affirmative ballot.

- (4) If any objection cannot be resolved, the entire ballot remains negative.
- (5) Once more than seventy-five percent of the ballots received (that had voted either affirmative or negative) have been turned affirmative, two lists are published to the entire balloting group: the detailed list of approved changes and the list of unresolved objections, along with our reasons for rejecting them. This is known as a *recirculation*. You have minimum of ten days (after an appropriate time to ensure the mail got through) to review these two lists and take one of the following actions:
 - (a) Do nothing; your ballots will continue to be counted as we have classified them, based on items (3) and (4).
 - (b) Explicitly change your negative ballot to affirmative by agreeing to remove all of your objections from the unresolved list.
 - (c) Explicitly change your affirmative ballot to negative based on your disapproval of either of the two lists you reviewed. If an issue is not on one of the two lists, new objections about this are not allowed. Negative ballots that come in on recirculations cannot be cumulative. They shall repeat any objections that the balloter considers unresolved from the previous recirculation. Ballots that simply say “and all the unresolved objections from last time” will be declared unresponsive. Ballots that are silent will be presumed to fully replace the previous ballot, and all objections not mentioned on the most current ballot will be considered as successfully resolved.
- (6) The list of changes will frequently be a new draft document with the changes integrated. This is not a requirement, however, and a small number of changes may prompt merely a change list approach to recirculation.
- (7) A copy of all your objections and our resolutions will be mailed to you. You can receive the full package of all resolutions from all ballots by contacting the IEEE Standards Office (who will probably charge you for the copying involved). If you don’t agree with one of our resolutions and haven’t been contacted personally before you receive this list, please accept our apologies and submit a new ballot against the new draft during the recirculation period.
- (8) If at the end of the recirculation period there remain greater than seventy-five percent affirmative ballots, and no new objections have been received, a new draft is prepared that incorporates all the changes. This draft and the unresolved objections list go to the IEEE Standards Board for approval. If the changes cause too many ballots to slip back into negative status, another resolution and recirculation cycle begins.

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

Balloting Guidelines

This section consists of guidelines on how to write and submit the most effective ballot possible. The activity of resolving balloting comments is difficult and time consuming. Poorly constructed comments can make that even worse.

We have found several things that can be done to a ballot that make our job more difficult than it needs to be, and likely will result in a less than optimal response to ballots that do not follow the form below. Thus it is to your advantage, as well as ours, for you to follow these recommendations and requirements.

If a ballot that significantly violates the guidelines described in this section comes to us, we will determine that the ballot is unresponsive, and simply ignore all the material in it.

Secondly, objections that don't contain a specification so that the correction to resolve the objection "can be readily determined" are also unresponsive and will be ignored.

(If we do recognize a ballot that is generally "unresponsive," we will try to inform the balloter as soon as possible so he/she can correct it, but it is ultimately the balloter's responsibility to assure the ballot is responsive.)

Typesetting is not particularly useful to us. And please do not send handwritten ballots. Typewritten (or equivalent) is fine, and if some font information is lost it will be restored by the Technical Editor in any case. If you use `nroff`, you will include extraneous spacing and sometimes backspaces and overstrikes; if you really must use `nroff`, please turn off hyphenation and line adjusting:

```
.hy 0
.na
```

and run the output through `col -b` to remove all the overstrikes. (Also remember that backslashes and leading periods and apostrophes in your text will be treated impolitely by the `*roff` family). The ideal ballot is formatted as a "flat ASCII file," without any attempt at reproducing the typography of the draft and without embedded control characters or overstrikes; it is then printed in Courier (or some other typewriter-like) font for paper-mailing to the IEEE Standards Office and simultaneously e-mailed to the working group Chair.

Don't quote others' ballots. Cite them if you want to refer to another's ballot. If more than one person wants to endorse the same ballot, send just the cover sheets and one copy of the comments and objections. [Note to Institutional Representatives of groups like X/Open, OSF, UI, etc.: this applies to you, too. Please don't duplicate objection text with your members.] Multiple identical copies are easy to deal with, but just increase the paper volume. Multiple almost-identical ballots are a disaster, because we can't tell if they are identical or not, and are likely to miss the subtle differences. Responses of the forms:

— "I agree with the item in <someone>'s ballot, but I'd like to see this done instead"

— "I am familiar with the changes to `f00` in <someone>'s ballot and I would object if this change is [or is not] included"

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

are very useful information to us. If we resolve the objection with the original balloter (the one whose ballot you are referencing), we will also consider yours to be closed, unless you specifically include some text in your objection indicating that should not be done.

Be very careful of “Oh, by the way, this applies <here> too” items, particularly if they are in different sections of the document that are likely to be seen by different reviewers. They are probably going to be missed! Note the problem in the appropriate section, and cite the detailed description if it’s too much trouble to copy it. The reviewers don’t have time to read the whole ballot, and only read the parts that appear to apply to them. Particularly where definitions are involved, even if the change really belongs in one section but the relevant content is in another, an extra cross-reference would be indicated. If you wish to endorse someone else’s ballot, either in whole or part, be specific about whether you will be automatically satisfied if they are satisfied. If you will not necessarily be satisfied if they are, your ballot could be deemed unresponsive because it does not give achievable conditions under which your ballot could be converted to affirmative. You then must give the conditions under which you would be satisfied as well. If you would be satisfied in some areas and not in others, it is best to specifically point to each specific objection in the ballot you point to, giving the conditions for each.

Please consider this a new ballot that should stand on its own. Please do not make backward references to your ballots for previous drafts—include all the text you want considered here, because the Technical Reviewer may not have your old ballot. And, the old section and line numbers won’t match up anyway. If one of your objections was not accepted exactly as you wanted, it will not be useful to send in the exact text you sent before; read the nearby Rationale section and come up with a more compelling (or clearly-stated) justification for the change.

Please be very wary about global statements, such as “all of the arithmetic functions need to be defined more clearly.” Unless you are prepared to cite specific instances of where you want changes made, with reasonably precise replacement language, your ballot will be considered unresponsive.

Ballot Form

The following form is recommended. We would greatly appreciate it if you sent the ballot in electronic form in addition to the required paper copy. Our policy is to handle all ballots online, so if you don’t send it to us that way, we have to type it in manually. For the last POSIX.2 ballot, only one or two balloters could not accommodate us on this and thus we had very little typing to do. See the first page of this Annex for the addresses and media. As you’ll see from the following, formatting a ballot that’s sent to us online is much simpler than a paper-only ballot.

The ballot should be page-numbered, and contain the name, e-mail address, and phone number(s) of the objector(s). (If you send us only a paper copy, make sure this information appears on every page; electronic ballots just need it once, in the beginning.) The lines before the first dashed line are a page header, and should only appear once on each page. Please leave adequate (at least one inch) margins

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

on both sides. Each objection/comment/editorial comment should be sequentially numbered, not in individual ranges [i.e., not Objection #1, Comment #1]

Since we deal with the ballots online, there is no longer any requirement to put only one objection or section per page.

Don't format the ballot as a letter or document with its *own* section numbers. These are simply confusing. As shown below, it is best if you cause each objection and comment to have a sequential number that we can refer to amongst ourselves and to you over the phone. Number sequentially from 1 and count objections, comments, and editorial comments the same; don't number each in its own range. If you don't do this, we'll number them ourselves, but you won't know what numbers we're using.

Please precede each objection/comment with a little code line (if you don't, we'll have to do it ourselves):

```
@ <section>.<clause> <code> <seqno>
```

where:

- | | |
|-----------|--|
| @ | At-sign in column 1 (which means no @'s in any other column 1's). |
| <section> | The major section (chapter or annex) number or letter in column 3. Use zero for Global or for something, like the frontmatter, that has no section or annex number. |
| <clause> | The clause number (second-level header). Please do not go deeper than these two levels. In the text of your objection or comment, go as deep as you can in describing the location, but this code line uses two levels only. |
| <code> | One of the following lowercase letters, preceded and followed by spaces: <ul style="list-style-type: none"> o Objection. c Comment or Editorial Comment. |
| <seqno> | A sequence number, counting all objections and comments in a single range. |

Objection:

Balloter Name (202)555-1212 page x of nn.

E-Mail Address FAX: Fax Number

Balloter2 Name (303)555-1213

E-Mail Address2 FAX: Fax Number2

@ x.y o seq#

<Seq#> Sect x.y OBJECTION. page xxx, line zzz:

Problem:

A clear statement of the problem that is observed, sufficient for others to understand the nature of the problem. Note that you should identify problems by section, page, and line numbers. This may seem redundant, but if you transpose a digit pair, we may get totally lost without a cross-check like this. Use the line number where the problem starts, not just where the section itself starts; we sometimes attempt to sort objections by line numbers to make editing more accurate. If you are referring to a range of lines, please don't say "lines 1000ff;" use a real range so we can tell where to stop looking. If you have access to the online versions of a balloting draft, please do not send in a ballot that refers to the page numbers in the nroff output version; use only the line and page numbers found in the printed draft or the online PostScript draft. We will really love you if you can manage to include enough context information in the problem statement (such as the name of the utility) so we can understand it without having the draft in our laps at the time. (It also helps you when we e-mail it back to you.) If you are objecting to an action in the Unresolved Objections List, use the section/page/line number reference for the appropriate place in the standard; don't refer to the UOL except to cite its number and for clarification of your points.

Action:

A precise statement of the actions to be taken on the document to resolve the objection above, which if taken verbatim will completely remove the objection.

If there is an acceptable range of actions, any of which will resolve the problem for you if taken exactly, please indicate all of them. If we accept any of these, your objection will be considered as resolved.

If the Action section is omitted or is vague in its solution, the objection will be reclassified as a nonbinding comment. The Technical Reviewers, being human, will give more attention to Actions that are well-described than ones that are vague or imprecise. The best ballots of all have very explicit directions to substitute, delete, or add text in a style consistent with the rest of the document, such as:

```

316 Delete the sentence on lines 101-102:
317     "The implementation shall not ... or standard error."
318 On line 245, change "shall not" to "should not".
319 After line 103, add:
320     -r      Reverse the order of bytes read from the file.
321 Some examples of poorly-constructed actions:
322 Remove all features of this command that are not supported by BSD.
323 Add -i.
324 Make this command more efficient and reliable.
325 Use some other flag that isn't so confusing.
326 I don't understand this section.
327 Specify a value--I don't care what.
328 Objection Example:
329 Hal Jespersen                (415) 364-3410                page 3 of 17.
330 UUCP: hlj@Posix.COM  FAX:    (415) 364-4498
331 -----
332 @ 2.6 o 23
333 23. Sect 2.6 OBJECTION. page 77, line 1217:
334 Problem:
335 The EDITOR environment variable is not used as stated
336 in my company. This description would cause hundreds
337 of my shell scripts to break.
338 Action:
339 Change the first sentence on line 1217 to:
340     The e-mail address of the editor of the user's
341     favorite POSIX standard.
342 -----
343 @ 3.1 o 24
344 24. Sect 3.1.6 OBJECTION. page 123, line 17:
345 Problem:
346 I support UO 3.01-999-6 concerning the objection to the      1
347 definition of "operator".                                     1
348 This definition would cause great hardship to the users      1
349 of the systems I develop.                                     1
350 I feel your rationale for rejection was inappropriate         1
351 because you overlooked the following technical points [etc.]... 1
352 Action:
353 Change the term "operator" to "operation-symbol" in this
354 definition and globally throughout Section 3.
355 Comment:

```

 @ x.z c seq#
 <Seq#> Sect x.z COMMENT. page xxx, line zzz:

A statement of a problem that you might want to be resolved by the reviewer, but which does not in any way affect whether your ballot is negative or positive. The form for objections is not required, but it increases the probability that your comment will have an effect on the final document.

Although there may be questions to you or responses on the topic, no changes in the drafts are required by a comment, although it will be looked at to determine whether the concern should be addressed. It is possible to abuse this rule and label all of your comments as objections, but it is a significant disservice to the individuals who are volunteering their time to address your concerns.

Remember that any issue concerning the pages preceding page 1 (the Frontmatter), Rationale text with shaded margins, Annexes, NOTES in the text, footnotes, or examples will be treated as a nonbinding comment whether you label it that way or not, but it would help us if you'd label it correctly.

Editorial Comment:

 @ x.z c seq#
 <Seq#> Sect x.z EDITORIAL COMMENT. page xxx, line zzz:

These are for strictly editorial issues, where the technical meaning of the document is not changed. Examples are: typos; misspellings; English syntax or usage errors; appearances of lists or tables; arrangement of sections, clauses, and sub-clauses (except where the location of information changes the optionality of a feature). Marking these as comments but indicating that they are editorial speeds the process.

Please be aware that after balloting concludes the document will be subjected to more sets of editors at the IEEE and ISO who are empowered to make broad editorial changes and rewording (for example, to get the text ready for translation into French.)

Thank you for your cooperation in this important balloting process.

Hal Jespersen

Identifier Index

[test — Evaluate expression {4.62}	631
ar	ar — Create and maintain library archives {6.1}	687
asa	asa — Interpret carriage-control characters {C.1}	813
awk	awk — Pattern scanning and processing language {4.1}	263
basename	basename — Return nondirectory portion of pathname {4.2}	297
bc	bc — Arbitrary-precision arithmetic language {4.3}	301
break	break — Exit from for , while , or until loop {3.14.1}	246
c89	c89 — Compile Standard C programs {A.1}	726
case	case Conditional Construct {3.9.4.3}	227
cat	cat — Concatenate and print files {4.4}	318
cd	cd — Change working directory {4.5}	322
chgrp	chgrp — Change file group ownership {4.6}	326
chmod	chmod — Change file modes {4.7}	329
chown	chown — Change file ownership {4.8}	337
cksum	cksum — Write file checksums and sizes {4.9}	341
cmp	cmp — Compare two files {4.10}	347
colon	colon — Null utility {3.14.2}	247
comm	comm — Select or reject lines common to two files {4.11} ...	350
command	command — Execute a simple command {4.12}	354
<i>confstr()</i>	C Binding for Get String-Valued Configurable Variables {B.10.1}	809
continue	continue — Continue for , while , or until loop {3.14.3}	248
cp	cp — Copy files {4.13}	359
cut	cut — Cut out selected fields of each line of a file {4.14} ..	368
date	date — Write the date and time {4.15}	373
dd	dd — Convert and copy a file {4.16}	379
diff	diff — Compare two files {4.17}	388
dirname	dirname — Return directory portion of pathname {4.18}	395
dot	dot — Execute commands in current environment {3.14.4}	248
echo	echo — Write arguments to standard output {4.19}	399
ed	ed — Edit text {4.20}	402
env	env — Set environment for command invocation {4.21}	419
eval	eval — Construct command by concatenating arguments {3.14.5}	249
exec	exec — Execute commands and open, close, and/or copy file descriptors {3.14.6}	250
exit	exit — Cause the shell to exit {3.14.7}	251
export	export — Set export attribute for variables {3.14.8}	252
expr	expr — Evaluate arguments as an expression {4.22}	423

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

<code>false</code>	false — Return false value {4.23}	428
<code>find</code>	find — Find files {4.24}	430
<code>fnmatch()</code>	C Binding for Match Filename or Pathname {B.6}	794
<code>fold</code>	fold — Fold lines {4.25}	438
<code>for</code>	for Loop {3.9.4.2}	226
<code>fort77</code>	fort77 — FORTRAN compiler {C.2}	817
<code>fpathconf()</code>	C Binding for Get Numeric-Valued Configurable Variables {B.10.2}	811
<code>getconf</code>	getconf — Get configuration values {4.26}	442
<code>getenv()</code>	C Binding for Access Environment Variables {B.4}	786
<code>getopt()</code>	C Binding for Command Option Parsing {B.7}	796
<code>getopts</code>	getopts — Parse utility options {4.27}	447
<code>glob()</code>	C Binding for Generate Pathnames Matching a Pattern {B.8}	799
<code>glob_t</code>	Description {B.8.2}	799
<code>grep</code>	grep — File pattern searcher {4.28}	452
<code>head</code>	head — Copy the first part of files {4.29}	459
<code>id</code>	id — Return user identity {4.30}	462
<code>if</code>	if Conditional Construct {3.9.4.4}	228
<code>join</code>	join — Relational database operator {4.31}	466
<code>kill</code>	kill — Terminate or signal processes {4.32}	471
<code>lex</code>	lex — Generate programs for lexical tasks {A.2}	736
<code>ln</code>	ln — Link files {4.33}	476
<code>locale</code>	locale — Get locale-specific information {4.34}	480
<code>localedef</code>	localedef — Define locale environment {4.35}	486
<code>logger</code>	logger — Log messages {4.36}	491
<code>logname</code>	logname — Return user's login name {4.37}	494
<code>lp</code>	lp — Send files to a printer {4.38}	496
<code>ls</code>	ls — List directory contents {4.39}	502
<code>mailx</code>	mailx — Process messages {4.40}	510
<code>make</code>	make — Maintain, update, and regenerate groups of pro- grams {6.2}	695
<code>mkdir</code>	mkdir — Make directories {4.41}	514
<code>mkfifo</code>	mkfifo — Make FIFO special files {4.42}	518
<code>mv</code>	mv — Move files {4.43}	521
<code>nohup</code>	nohup — Invoke a utility immune to hangups {4.44}	526
<code>od</code>	od — Dump files in various formats {4.45}	530
<code>paste</code>	paste — Merge corresponding or subsequent lines of files {4.46}	538
<code>pathchk</code>	pathchk — Check pathnames {4.47}	543
<code>pathconf()</code>	C Binding for Get Numeric-Valued Configurable Variables {B.10.2}	811
<code>pax</code>	pax — Portable archive interchange {4.48}	548
<code>pclose()</code>	C Binding for Pipe Communications with Programs {B.3.2}	782
<code>popen()</code>	C Binding for Pipe Communications with Programs {B.3.2}	782
<code>pr</code>	pr — Print files {4.49}	562

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

<code>printf</code>	printf — Write formatted output {4.50}	568
<code>pwd</code>	pwd — Return working directory name {4.51}	574
<code>read</code>	read — Read a line from standard input {4.52}	576
<code>readonly</code>	readonly — Set read-only attribute for variables {3.14.9}	253
<code>regcomp()</code>	C Binding for Regular Expression Matching {B.5}	786
<code>regerror()</code>	C Binding for Regular Expression Matching {B.5}	786
<code>regexexec()</code>	C Binding for Regular Expression Matching {B.5}	786
<code>regex_t</code>	Description {B.5.2}	786
<code>regfree()</code>	C Binding for Regular Expression Matching {B.5}	786
<code>regmatch_t</code>	Description {B.5.2}	786
<code>regoff_t</code>	Description {B.5.2}	786
<code>return</code>	return — Return from a function {3.14.10}	254
<code>rm</code>	rm — Remove directory entries {4.53}	579
<code>rmdir</code>	rmdir — Remove directories {4.54}	584
<code>sed</code>	sed — Stream editor {4.55}	587
<code>set</code>	set — Set/unset options and positional parameters {3.14.11}	254
<code>sh</code>	sh — Shell, the standard command language interpreter {4.56}	597
<code>shift</code>	shift — Shift positional parameters {3.14.12}	258
<code>sleep</code>	sleep — Suspend execution for an interval {4.57}	603
<code>sort</code>	sort — Sort, merge, or sequence check text files {4.58}	605
<code>strip</code>	strip — Remove unnecessary information from execut- able files {6.3}	716
<code>stty</code>	stty — Set the options for a terminal {4.59}	613
<code>sysconf()</code>	C Binding for Get Numeric-Valued Configurable Variables {B.10.2}	811
<code>system()</code>	C Binding for Execute Command {B.3.1}	778
<code>tail</code>	tail — Copy the last part of a file {4.60}	623
<code>tee</code>	tee — Duplicate standard input {4.61}	628
<code>test</code>	test — Evaluate expression {4.62}	631
<code>touch</code>	touch — Change file access and modification times {4.63}	640
<code>tr</code>	tr — Translate characters {4.64}	645
<code>trap</code>	trap — Trap signals {3.14.13}	258
<code>true</code>	true — Return true value {4.65}	652
<code>tty</code>	tty — Return user's terminal name {4.66}	654
<code>umask</code>	umask — Get or set the file mode creation mask {4.67}	657
<code>uname</code>	uname — Return system name {4.68}	662
<code>uniq</code>	uniq — Report or filter out repeated lines in a file {4.69}	665
<code>unset</code>	unset — Unset values and attributes of variables and functions {3.14.14}	260
<code>wait</code>	wait — Await process completion {4.70}	669
<code>wc</code>	wc — Word, line, and byte count {4.71}	674
<code>while</code>	until Loop {3.9.4.6}	229
<code>while</code>	while Loop {3.9.4.5}	229

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

<i>wordexp()</i>	C Binding for Perform Word Expansions {B.9}	804
<i>wordexp_t</i>	Description {B.9.2}	804
<i>xargs</i>	xargs — Construct argument list(s) and invoke utility {4.72}	678
<i>yacc</i>	yacc — Yet another compiler compiler {A.3}	750

Alphabetic Topical Index

A

- A/2047 ... 597
- / ... 109
- // ... 297, 300, 395, 398
- [
 - definition of ... 631
- Abbreviations ... 50
- ABCDEF ... 170, 742
- ABC ... 572
- absolute pathname
 - definition of ... 26
- access control
 - additional ... 361
 - alternate ... 361
- Access Environment Variables ... 720, 844
- ACK ... 68, 74
- ACK ... 56, 618
- ACM ... 346
- Actions Equivalent to POSIX.1 Functions
 - ... 144
- Actions ... 275
- {ACTSIZE} ... 764
- ACUTE ... 100-101
- adb ... 833
- ADD_ASSIGN ... 282, 286-287
- address space
 - definition of ... 26
- ADD_ASSIGN ... 290
- affirmative response ... 26, 92, 104, 360, 362-363, 432, 434, 485, 487, 521, 524, 555, 579-581
 - definition of ... 26
- A-F ... 317
- Aim of Character Mnemonics ... 884
- <alert>
 - definition of ... 26
- Algorithms ... 764
- Allow Historical Conforming Applications
 - ... 6
- AM/PM ... 89
- AM/PM ... 374
- AND_IF ... 235-236
- AND ... 282, 285-287
- AND Lists ... 225
- AND/OR lists ... 218, 222, 225
- AND-OR ... 216
- AND/OR ... 218
- AND-OR ... 222
- AND ... 146, 222, 225, 254-255, 268-269, 290, 296, 433, 525, 583
- angle brackets
 - definition of ... 26
- ANSI ... 826
- a.out ... 726-727, 732, 735, 816-818, 822-823
- Append Command ... 408
- Appending Redirected Output ... 210
- APPEND ... 282, 284
- APPEND ... 290
- Application Conformance ... 15
- application
 - definition of ... 48
- apply ... 681
- appropriate privileges ... 26, 32-33, 41, 329, 336, 462, 486-487, 557, 559, 561
 - definition of ... 26
- ar
 - Create and maintain library archives
 - ... 687, 843
 - ... 687-690, 692-694, 728, 732, 818-819
 - definition of ... 687
- ARFLAGS ... 707-708
- ARGC-1 ... 292
- ARGC
 - awk variable ... 271
- ARGC ... 265, 271, 292-293
- {ARG_MAX} ... 175-176, 179, 185, 216, 678, 682-683
- {ARG_MAX} ... 723
- Argument Processing with *getopt()* ... 798
- argument
 - definition of ... 26
- argv ... 26

ARGV
 awk variable ... 271
 ARGV ... 265, 271, 292-293
 Arithmetic Expansion ... 205
 Arithmetic Functions ... 278
 Arithmetic Precision and Operations ... 145
 ARPANET ... 513
 asa
 — Interpret carriage control characters
 ... 846
 — Interpret carriage-control characters
 ... 813
 ... 813-816
 definition of ... 813
 ASCII ... 74, 102
 ASCII to EBCDIC Conversion ... 385
 ASCII to IBM EBCDIC Conversion ... 386
 ASCII ... 48-49, 58-59, 129-130, 132, 385-387,
 536-538, 621, 693, 816
 ASSIGNMENT ... 233
 ASSIGNMENT_WORD ... 234
 ASSIGNMENT_WORD ... 235, 237
 ASSIGN_OP ... 303, 305
 ASSIGN_OP ... 22, 306
 asterisk
 definition of ... 26
 Asynchronous Lists ... 224
 at ... 833
 AT&T ... 9-10, 557, 572
 awk
 — Pattern scanning and processing
 language ... 263, 838
 ... 5, 25, 41, 44, 60, 129, 144, 155, 158,
 161, 163, 165, 172, 178, 208, 263-267,
 270-275, 277, 279, 281, 288, 290-291,
 293-296, 317, 573, 748
 Arithmetic Functions ... 278
 definition of ... 263
 Escape Sequences ... 289
 Expressions ... 267
 Expressions in Decreasing Precedence
 ... 268
 Functions ... 277
 Grammar ... 281
 Input/Output and General Functions
 ... 280
 Lexical Conventions ... 288
 Output Statements ... 276
 Patterns ... 274
 Regular Expressions ... 273
 String Functions ... 278
 User-Defined Functions ... 281
 Variables and Special Variables ... 271

AWK ... 9, 293-294, 826
 A-Z ... 650

B

B.3 ... 784
 background process group
 definition of ... 27
 background process
 definition of ... 26
 background ... 26-27, 36
 background ... 26-27, 35, 129, 159, 163, 193,
 221, 224, 476, 617, 672-673, 829
 backquote
 definition of ... 27
 BACKREF ... 122-123
 backslash
 definition of ... 27
 <backspace>
 definition of ... 27
 Balloting Instructions ... 919
 basename
 — Return nondirectory portion of pathname
 ... 297, 838
 ... 298-300, 398
 basename
 definition of ... 27
 basename
 definition of ... 297
 basic regular expression
 definition of ... 27
 Basic Regular Expressions ... 112
 bc
 — Arbitrary-precision arithmetic language
 ... 301, 838
 ... 25, 44, 155, 173-174, 301-303, 305,
 308, 311-317, 573, 834
 definition of ... 301
 Grammar ... 303
 Lexical Conventions ... 305
 Operations ... 307
 Operators ... 307
 {BC_BASE_MAX} ... 176, 308
 BC_BASE_MAX ... 173-174
 {BC_BASE_MAX} ... 174
 BC_BASE_MAX ... 775
 {BC_BASE_MAX} ... 775, 811
 {BC_DIM_MAX} ... 175-176, 307

- BC_DIM_MAX ... 173-174
 - {BC_DIM_MAX} ... 174
 - BC_DIM_MAX ... 775
 - {BC_DIM_MAX} ... 775, 811
 - {BC_SCALE_MAX} ... 175-176, 307
 - BC_SCALE_MAX ... 173-174
 - {BC_SCALE_MAX} ... 174
 - BC_SCALE_MAX ... 775
 - {BC_SCALE_MAX} ... 775, 811
 - {BC_STRING_MAX} ... 305
 - BC_STRING_MAX ... 173-174
 - {BC_STRING_MAX} ... 174
 - BC_STRING_MAX ... 775
 - {BC_STRING_MAX} ... 775, 811
 - BEGIN ... 282
 - BEGIN ... 264-265, 267, 272, 274-275, 289, 292, 294-295, 744
 - BEL ... 56, 618
 - Bibliography ... 825
 - /bin ... 109
 - blank line
 - definition of ... 27
 - <blank>
 - definition of ... 27
 - block special file ... 351
 - definition of ... 27
 - BNF ... 10
 - BODY ... 756
 - braces
 - definition of ... 27
 - bracket expression ... 113
 - brackets
 - definition of ... 28
 - break ... 191, 228, 247, 275
 - definition of ... 246
 - BRE [ERE] matching a single character
 - definition of ... 111
 - BRE [ERE] matching multiple characters
 - definition of ... 111
 - BRE Expression Anchoring ... 116
 - BRE Ordinary Characters ... 112
 - BRE Precedence ... 116-117
 - BRE Special Characters ... 112
 - BRE
 - abbreviation ... 50
 - BRE/ERE Grammar Lexical Conventions ... 121
 - BRE/ERE ... 121
 - BRE ... 110-113, 115-117, 127-128, 136, 453-454
 - BREs Matching a Single Character or Collating Element ... 112
 - BREs Matching Multiple Characters ... 115
 - BRKINT ... 615
 - BSD/32V ... 596
 - BSD
 - 4.2 ... 387
 - 4.3 ... 315, 336, 340, 365, 393-394, 397, 436, 525, 529
 - 4.4 ... 693
 - BSD ... 4-5, 9, 47, 181, 187, 213, 314, 320-321, 325, 328, 335-336, 339-340, 345, 365, 378, 387, 394, 401-402, 416-418, 437-438, 458, 462, 465-466, 470, 479, 493, 501, 508-510, 514, 525, 536-538, 557-559, 567, 595-596, 621, 626-627, 630, 635-639, 644-645, 650-651, 681, 694, 708-710, 712, 714, 734, 749, 832, 837
 - BUFSIZ ... 177
 - Built-in Utilities ... 51, 830
 - built-in utility
 - definition of ... 28
 - builtin ... 357
 - BUILTIN_FUNC_NAME ... 282, 286-287
 - built-in ... 21, 28, 30, 36, 45-46, 51-52, 176, 181-182, 192, 214, 217, 219-221, 226, 230-231, 241, 246, 248, 252-253, 255, 259, 261, 263, 273-274, 277, 282, 289-290, 293, 325, 354, 356
 - builtin ... 357
 - built-in ... 358, 419, 422, 431, 450, 452, 476, 527, 529, 547, 576, 578-579, 601-602, 653, 660, 672, 679, 695-696, 699, 703-704, 711-712, 830, 832
 - BUILTIN_FUNC_NAME ... 289
 - byte
 - definition of ... 28
- ## C
- C Binding for Access Environment Variables ... 786, 845
 - C Binding for Command Option Parsing ... 796, 845
 - C Binding for Execute Command ... 778
 - C Binding for Generate Pathnames Matching a Pattern ... 799, 845

- C Binding for Get Numeric-Valued Configurable Variables ... 811
- C Binding for Get POSIX Configurable Variables ... 809, 845
- C Binding for Get String-Valued Configurable Variables ... 809
- C Binding for Locale Control ... 812, 846
- C Binding for Match Filename or Pathname ... 794, 845
- C Binding for Perform Word Expansions ... 804, 845
- C Binding for Pipe Communications with Programs ... 782
- C Binding for Regular Expression Matching ... 786, 845
- C Binding for Shell Command Interface ... 778, 845
- C Bindings for Numeric-Valued Configurable Variables ... 811
- C Compile-Time Symbolic Constants ... 776
- C Execution-Time Symbolic Constants ... 777
- C Language Bindings Option ... 771, 845
- C Language Definitions ... 772, 845
- C Language Development Utilities Option ... 725, 844
- C Macros for Symbolic Limits ... 775
- C Numerical Limits ... 775, 845
- C Shell ... 181, 202, 232, 245, 325, 475, 637
- C Standard Operators and Functions ... 146
- C Standard ... 3, 22, 28, 44-46, 48-50, 62, 71, 81, 86-87, 106, 129-131, 134, 144-146, 150, 154, 171, 175, 177, 206, 267-270, 275, 278, 288, 293-294, 296, 321, 378, 537, 547, 570, 572-573, 630, 677, 726, 733, 736, 740, 748, 750, 754, 772-774, 778-780, 782, 792-793
definition of ... 50
- c89
 - Compile Standard C programs ... 726, 845
 - ... 4, 154, 533, 537, 712, 726-727, 729-735, 739, 745, 750, 753, 763-764, 766, 822, 824, 833, 835
 - definition of ... 726
- can
 - definition of ... 23
- CAN ... 68, 74
- CAN ... 56, 618
- carriage-control characters ... 814
- <carriage-return>
 - definition of ... 28
- case conversion ... 67
- case ... 191, 206, 216, 227-228, 234, 239, 243
 - Conditional Construct ... 227
 - definition of ... 227
- cat
 - Concatenate and print files ... 318, 838
 - ... 158, 251, 318-322, 595, 630
 - definition of ... 318
- C_BIND ... 179, 446, 730
- cc ... 154, 712, 733-734
- CCITT ... 513
- CC**
 - variable ... 162
- cd
 - Change working directory ... 322, 838
 - ... 51-52, 107, 143, 220, 240-241, 322-325, 356
 - definition of ... 322
- C_DEV ... 179-180, 777, 811
- CDPATH**
 - variable ... 107, 323-325
- CFLAGS ... 706-708, 711, 713
- Change Command ... 408
- Changing the Current Working Directory ... 143
- character attributes ... 67
- character case conversion ... 67
- character class expression ... 114
- character class
 - definition of ... 29
- character classification ... 67
- Character Mnemonics Classes ... 885
- Character Mnemonics Guidelines ... 884
- Character Set and Symbolic Names ... 54
- Character Set Description File ... 55
- Character Set ... 54, 830
- character set
 - definition of ... 48
 - portable ... 54
- character special file ... 351
 - definition of ... 29
- character
 - definition of ... 28
- charmap file ... 55-56, 58-60, 65, 73, 77, 99, 481, 483-484, 486-490, 621, 847, 850, 887, 912

- CHARMAP ... 56, 99
- CHAR ... 94
- {CHAR_BIT} ... 46, 418
- {CHAR_MAX} ... 86-87
- CHARSET**
 - variable ... 849
- CHARSYMBOL ... 94
- CHARSYMBOL ... 93
- chdir()* ... 143
- chgrp
 - Change file group ownership ... 326, 839
 - ... 326-328, 340
 - definition of ... 326
- {CHILD_MAX} ... 176, 224, 672
- {CHILD_MAX} ... 723
- chmod
 - Change file modes ... 329, 839
 - ... 3, 5, 329-330, 334-336, 436, 508, 515, 517-518, 657-658, 661
 - definition of ... 329
 - Grammar ... 333
- chmod()* ... 3, 32-33, 141, 334, 336
- chown
 - Change file ownership ... 337, 839
 - ... 337-340
 - definition of ... 337
- chown()* ... 326, 337, 339-340
- CH-1211 ... 12, 825
- C_IDENTIFIER ... 759-760, 768
- circumflex
 - definition of ... 29
- cksum
 - Write file checksums and block counts ... 839
 - Write file checksums and sizes ... 341
 - ... 5, 341-343, 345-346, 709, 837
 - definition of ... 341
- Clean Up the Interfaces ... 4
- {CLK_TCK} ... 723
- CLK_TCK ... 443
- CLOBBER ... 235, 238
- CLOCAL ... 615
- cmp
 - Compare two files ... 347, 839
 - ... 347-350
 - definition of ... 347
- Code file ... 752
- col ... 833
- collating element
 - definition of ... 29
- collating symbol ... 114
- collating-element
 - Keyword ... 73
- collating-symbol
 - Keyword ... 77
- Collation Order ... 78
- collation sequence
 - definition of ... 29
- collation sequences
 - defining ... 72
- collation
 - definition of ... 29
- COLLELEMENT ... 94-96
- COLLELEMENT ... 93
- COLL_ELEM ... 122, 124
- {COLL_WEIGHTS_MAX} ... 29, 72, 77, 79, 177, 489, 847
- COLL_WEIGHTS_MAX ... 173-174
- {COLL_WEIGHTS_MAX} ... 174
- COLL_WEIGHTS_MAX ... 775
- {COLL_WEIGHTS_MAX} ... 775, 811
- COLLSYMBOL ... 94-96
- colon — Null utility ... 247
- colon
 - definition of ... 247
- column position
 - definition of ... 30
- COLUMNS**
 - variable ... 107, 503-504
- Combination Modes ... 618
- comm
 - Select or reject lines common to two files ... 350, 839
 - ... 6, 350-353
 - definition of ... 350
- command language interpreter
 - definition of ... 30
- Command Option Parsing ... 721, 844
- Command Search and Execution ... 219
- Command Substitution ... 203
- command
 - Execute a simple command ... 354
 - Select or reject lines common to two files ... 839
 - ... 51-52, 107, 182, 215, 220, 354-358, 422, 529, 682

- command
 - definition of ... 30
- command
 - definition of ... 354
- command.c ... 713
- COMMENT_CHAR ... 711
- compile C programs ... 726
- compile FORTRAN programs ... 817
- Compile-Time Symbolic Constants for Portability Specifications ... 776
- Completing the Program ... 763
- Compound Commands ... 226
- Concepts Derived from the C Standard ... 144
- Concurrent Execution of Processes ... 139
- Configuration Values ... 173, 831
- Conflicts ... 761
- conformance document
 - definition of ... 23
- Conformance ... 12, 830
- conformance ... 2-3, 11-17, 23-24, 53, 107, 138, 153, 159-160, 317, 417, 461, 501, 626, 687, 719, 725, 771, 813, 826, 830, 843-844
- conforming application ... 6, 103, 128, 147, 149, 161, 166, 182, 598, 650, 693, 726, 733, 747, 817
- Conforming Implementation Options ... 14
- Conforming POSIX.2 Application Using Extensions ... 16
- Conforming POSIX.2 Application ... 15
- confstr() ... 443, 445, 722, 781, 793, 809-810
 - definition of ... 809
- confstr()
 - name Values ... 809
- Consequences of Shell Errors ... 214
- continue ... 247-248, 275
 - definition of ... 248
- Control Character Set ... 56
- Control Modes ... 614
- control operator
 - definition of ... 183
- controlling terminal ... 138
- Conventions ... 19, 830
- CONVFMT
 - awk variable ... 271
- CONVFMT ... 269, 276, 295
- Coordinated Universal Time (UTC) ... 378
- Copy Command ... 414
- core ... 713
- Covered Coded Character Sets ... 884
- cp
 - Copy files ... 359, 839
 - ... 359-367, 480, 525, 558, 560, 584
 - definition of ... 359
- cpio ... 66, 552, 557-561, 692
- CRC ... 341, 343, 345-346
- CREAD ... 615
- creat() ... 32, 387, 559, 640
- create ... 547
- cron ... 833
- CSA ... 99
- csh ... 529
- CSMA/CD ... 825
- csplit ... 833-834
- _CS_PATH ... 107, 356-357, 443, 809
- CS_PATH ... 107, 356-357, 443, 809
- CSTOPB ... 615
- csysconf() ... 810
- <ctype.h> ... 730
- current working directory ... 138
 - definition of ... 30, 46
- cut
 - Cut out selected fields of each line of a file ... 368, 839
 - ... 176, 179, 368, 370-372, 441, 541-542
 - definition of ... 368
- {CUT_FIELD_MAX} ... 176
- {CUT_LINE_MAX} ... 176
- C_VERSION ... 774, 776

D

- da_DK
 - (Example) Danish National Locale ... 850
- Danish Locale Model ... 848
- date formats ... 88
- date
 - Write the date and time ... 373, 839
 - ... 5, 88, 91, 106, 257, 373, 375-378, 505, 691
 - definition of ... 373
- DATE ... 377
- DBL_MANT_DIG ... 537

- DC1 ... 68, 74
- DC2 ... 68, 74
- DC3 ... 68, 74
- DC4 ... 68, 74
- dc ... 315-316, 834
- DC1 ... 56, 618
- DC2 ... 56, 618
- DC3 ... 56, 618
- DC4 ... 56, 618
- dd
 - Convert and copy a file ... 379, 839
 - ... 5, 379-387, 559, 833, 837
 - definition of ... 379
- DEAD**
 - variable ... 107, 511, 514
- Debugging the Parser ... 764
- DECIMAL_CHAR ... 94
- Declarations Section ... 754
- DECR ... 282, 286-287
- DECR ... 290
- Default Rules ... 706
- DEFAULT ... 706
- Definitions ... 23, 830
- Delete Command ... 408
- DEL ... 68, 76
- DEL ... 56, 618
- Dependencies on Other Standards ... 138, 830
- {DEPTH_MAX} ... 176-178
- /dev ... 109
- /dev/null ... 109-110, 214, 224, 458, 511, 514, 805-806, 808
- /dev/tty ... 109-110, 139, 213, 550, 554, 561, 567
- DGREAT ... 235, 238
- DIAERESIS ... 100
- diff
 - Compare two files ... 388, 839
 - ... 5, 388-395, 682, 836
 - c or C Output Format ... 391
 - Default Output Format ... 390
 - definition of ... 388
 - Directory Comparison Format ... 390
 - e Output Format ... 391
- DIGIT ... 746
- {DIGIT} ... 746
- directory entry ... 30-31, 33, 36-38, 40, 141, 165, 476-477, 502, 579, 581, 583-586, 841
 - definition of ... 30
- directory
 - current working ... 30
 - definition of ... 30
 - empty ... 31
 - parent ... 38
 - root ... 42
 - working ... 46
- directory ... 19-20, 30-34, 36, 38-39, 42, 46-47, 52, 103-105, 107, 109, 138-143, 155, 158-159, 164, 167, 176, 194-195, 197-198, 218, 221, 231, 240-241, 245, 248-249, 260, 297, 299-300, 322-326, 328-329, 331, 334-337, 339-340, 356-357, 359-360, 363-367, 371, 388, 390, 393-395, 398, 403-404, 430-431, 433, 436, 438, 445-446, 461, 476-477, 479-480, 502, 504, 506-509, 511, 514-517, 520-523, 525-527, 541, 543, 545, 547-550, 553, 557-561, 566, 574-575, 579-580, 582-585, 587, 598-599, 612, 631-632, 637, 639, 688, 698, 706, 726-727, 729, 733, 750, 767, 780, 795, 799-801, 810, 817-818, 820, 822, 834, 838-839, 841-842, 847
- dirname
 - Return directory portion of pathname
 - ... 395, 839
 - ... 300, 396-398
 - definition of ... 395
 - Examples ... 398
- DIS ... 884
- DIV_ASSIGN ... 282, 286-287
- DIV_ASSIGN ... 290
- DK-2900 ... 847
- DLE ... 68, 74
- DLE ... 56, 618
- DLESSDASH ... 235, 238
- DLESS ... 235, 238
- do ... 226-227, 296
- document
 - conformance ... 23
- Documentation ... 13
- documentation
 - system ... 24
- document ... 1-3, 9-16, 21, 23-25, 35, 49, 55, 60, 63, 66, 125, 149, 152-153, 159-161, 168, 177, 182, 185, 188-189, 201, 203, 211, 213-214, 218, 228, 234, 239-240, 243, 257, 294, 321, 336, 353, 387-388, 417, 427-428, 437, 458, 470, 475, 480, 496, 501, 595-596, 611-613, 627, 639, 677, 693, 725, 733-734, 740, 747, 764, 768-769, 774, 797, 801, 822-823, 825-826, 836

dollar-sign
 definition of ... 30
 done ... 226-227
 dot
 — Execute commands in current environ-
 ment ... 248
 dot ... 31, 34, 39
 dot ... 231, 248-249
 dot
 definition of ... 30
 dot
 definition of ... 248
 dot-dot ... 31, 34, 38-39
 definition of ... 30
 double-quote
 definition of ... 30
 Double-Quotes ... 186
 DOUBLE ... 707
 DSEMI ... 235, 237
 DUP_COUNT ... 122-125
 Duplicating an Input File Descriptor ... 211
 Duplicating an Output File Descriptor ... 212
 DUP_COUNT ... 121

E

E.4 ... 7, 843-844, 846
 [EACCES] ... 107
 EBCDIC ... 58-59, 385-387
 [ECHILD] ... 783-784
 echo
 — Write arguments to standard output
 ... 399, 839
 ... 5, 47, 188
 echo: ... 292
 echo ... 399-402, 493, 509, 572-573, 679,
 709, 715, 807
 definition of ... 399
 ECHOE ... 616
 ECHOK ... 617
 ECHONL ... 617
 ECHO ... 616, 744
 ECMA ... 885
 ed
 — Edit text ... 402, 839
 ... 6, 25, 44, 125-126, 141, 160-161,
 177, 278, 388, 391, 402-405, 407, 410,
 412, 416-418, 537, 551-552, 560, 595,
 748, 834
 Addresses ... 405

Commands ... 407
 definition of ... 402
 Regular Expressions ... 405
 ed.hup ... 404
 Edit Command ... 408
 Edit Without Checking Command ... 409
 Editorial Conventions ... 19
EDITOR
 variable ... 108
 {ED_FILE_MAX} ... 160, 177, 417
 {ED_LINE_MAX} ... 177, 417
 effective group ID ... 31, 33, 36, 42, 44, 138-
 140, 463, 465, 597
 definition of ... 31
 effective user ID ... 31, 34, 42, 45, 138-140,
 329, 463-465, 597
 definition of ... 31
 EFL ... 748
 egrep ... 125, 127, 453, 457, 747
 Eighth Edition UNIX ... 357
 [EINTR]
 EINTR ... 19, 781, 785
 [EINVAL] ... 783, 810
 elif ... 229
 ELLIPSIS ... 94-96
 else ... 229, 296
 empty directory
 definition of ... 31
 empty line
 definition of ... 31
 empty string
 definition of ... 31
 END ... 68, 76, 84, 88-89, 92, 95-99, 101-102,
 282
 END ... 56, 64, 99, 265, 267, 272, 274-275,
 289, 292, 294, 486
 [ENOENT] ... 357, 422, 529, 682
 [ENOEXEC] ... 107, 219-220
 ENQ ... 68, 74
 ENQ ... 56, 618
 entire regular expression
 definition of ... 110
 env
 — Set environment for command invocation
 ... 419, 839
 ... 51, 215, 357, 419-422, 485, 529, 682
 definition of ... 419
 ENVIRON
 awk variable ... 271

- ENVIRON ... 266, 271-272, 292-293
- ENV**
 - variable ... 195, 357
- EOF ... 303
- EOF ... 109, 307, 349, 617, 799
- EOL ... 94-99
- EOL ... 617
- EOT ... 68, 74
- EOT ... 56, 618
- [EPERM] ... 365
- Epoch ... 42
 - definition of ... 31
- equivalence class definition ... 73
- equivalence class expression ... 114
- equivalence class
 - definition of ... 31
- equivalence classes ... 72
- ERASE ... 27, 48, 616-618, 621
- ERE ... 125, 282, 286-287
- ERE Alternation ... 120
- ERE Bracket Expression ... 118
- ERE Expression Anchoring ... 120
- ERE Grammar ... 124
- ERE Ordinary Characters ... 118
- ERE Precedence ... 120
- ERE Special Characters ... 118
- ERE
 - abbreviation ... 50
 - definition of ... 50
- ERE ... 50, 110-111, 117-120, 124, 127, 136-137, 208, 263-264, 267, 273, 278-279, 288, 290, 294, 454, 458, 741-744, 747
- EREs Matching a Single Character or Collating Element ... 117
- EREs Matching Multiple Characters ... 119
- errfunc()* ... 801
- Error Handling ... 762
- Error Numbers ... 774
- ERR ... 260
- esac ... 227-228
- Escape Character (Backslash) ... 186
- Escape Sequences ... 169
- ESC ... 68, 74
- ESC ... 56, 618
- Establish the Locale ... 143
- ETB ... 68, 74
- ETB ... 56, 618
- /etc/Makefile ... 711
- ETX ... 68, 74
- ETX ... 56, 618
- eval
 - Construct command by concatenating arguments ... 249
 - ... 249
 - definition of ... 249
- ex ... 417, 791, 834
- Example Regular Expression Matching ... 791
- (Example)
 - Danish Charmap Files ... 887
 - Danish National Profile ... 847
- Examples ... 202
- [EXDEV] ... 521
- exec ... 217, 240-241, 250
- exec* ... 357, 422, 529, 682
- exec
 - definition of ... 250
- exec*
 - family ... 31, 42, 51, 107, 179, 221, 232, 246, 358, 678, 683, 778-779
- exec()* ... 251, 683
- execl()* ... 778
- execvp()* ... 422, 778
- executable file
 - definition of ... 31
- Execute Shell Command ... 720
- execute
 - definition of ... 31
- Execution Environment Utilities ... 263, 832
- Execution-Time Symbolic Constants for Portability Specifications ... 777
- execv()* ... 802
- execve()* ... 219-221, 714, 802
- execvp()* ... 422, 778, 802
- exit
 - Cause the shell to exit ... 251
 - ... 251, 254, 275, 290, 601
 - definition of ... 251
- exit()* ... 783
- EXIT ... 251, 258, 260, 546
- _exit()* ... 783
- expand
 - definition of ... 183
- export
 - Set export attribute for variables ... 252
 - ... 231, 240-241, 252-253

definition of ... 252

expr

- Evaluate arguments as an expression ... 423, 840
- ... 173-174, 206, 423-424, 426-427
- definition of ... 423
- Expressions ... 425

Expression Patterns ... 275

Expressions ... 267

{EXPR_NEST_MAX} ... 175, 177, 424

EXPR_NEST_MAX ... 173-174

{EXPR_NEST_MAX} ... 174

EXPR_NEST_MAX ... 775

{EXPR_NEST_MAX} ... 775, 811

extended regular expression

- definition of ... 32

Extended Regular Expressions ... 117

extended security controls

- definition of ... 32

EXTENDED_REG_EXP ... 94, 97

External Symbols ... 731, 821

F

f77 ... 822

false

- Return false value ... 428, 840
- ... 51, 53, 207, 428-429
- definition of ... 428

FAX ... 847

f() ... 732

FCEDIT

- variable ... 195

FD_CLOEXEC ... 780

feature test macro

- definition of ... 32

feature test macros ... 772

Features Inherited from POSIX.1 ... 138

FFLAGS ... 706-708

fgrep ... 453, 457, 459

fi ... 227

Field Splitting ... 207

field

- definition of ... 183

FIFO special file

- definition of ... 32

FIFO ... 32, 34, 40, 140, 347, 351, 361, 388, 433, 502, 506, 518, 520, 599, 624, 627, 631, 841

file access permissions ... 40

File Access Permissions ... 140

file access permissions

- definition of ... 32

File Contents ... 142

file descriptor ... 33, 38, 40, 138, 142, 209-213, 250-251, 360-362, 526, 584, 632, 734, 780, 782-783

- definition of ... 33

file descriptors ... 138

File Format Notation ... 168, 830

file group class

- definition of ... 33

file hierarchy

- definition of ... 33

file mode bits

- definition of ... 34

file mode creation mask ... 138

file mode

- definition of ... 34

file offset

- definition of ... 34

file other class

- definition of ... 34

file owner class

- definition of ... 34

file permission bits ... 32-34

- definition of ... 34

file permissions ... 31-34, 37, 44

File Removal ... 141

file serial number

- definition of ... 35

file system ... 7, 35, 41, 47, 51-52, 109, 141, 143-144, 159, 162, 166-167, 177, 182, 232, 357, 394, 437, 506-507, 509, 522, 525-526, 543, 552, 560-561, 632, 693-694, 834

- definition of ... 35
- read-only ... 41

File Time Values ... 142

file times update

- definition of ... 35

file type (see *file*)

file type

- definition of ... 35

file ... 834

file

- access permissions ... 32
- block special ... 27
- character special ... 29
- definition of ... 32
- FIFO special ... 32
- hierarchy ... 33

- locale definition ... 63
 - mode ... 34
 - offset ... 34
 - permission bits ... 34
 - regular ... 42
 - serial number ... 35
 - times update ... 35
 - File-Name Command ... 409
 - filename portability
 - definition of ... 34
 - filename
 - definition of ... 34
 - FILENAME
 - awk variable ... 272
 - FILE ... 745, 778, 782, 785
 - filter
 - definition of ... 35
 - find
 - Find files ... 430, 840
 - ... 5, 51, 142, 176-177, 190, 243, 336, 340, 394, 430-438, 559, 681-682, 795, 834
 - definition of ... 430
 - {FIND_DEPTH_MAX} ... 177
 - {FIND_FILESYS_MAX} ... 177
 - {FIND_NEWER_MAX} ... 177
 - flex ... 749
 - FLT_MANT_DIG ... 537
 - FMN_PATHNAME ... 795
 - fnmatch()* ... 243, 721, 793-795, 802
 - definition of ... 794
 - flags* Argument ... 794
 - <fnmatch.h> ... 794-795
 - FNM ... 773
 - FNM_NOESCAPE ... 794
 - FNM_NOMATCH ... 795
 - FNM_PATHNAME ... 794-795
 - FNM_PERIOD ... 794
 - FNR
 - awk variable ... 272
 - FNR ... 280, 293
 - fo_DK
 - (Example) Faroese LC_TIME and LC_MESSAGES ... 882
 - fold
 - Filter for folding lines ... 840
 - Fold lines ... 438
 - ... 178-179, 372, 438-442, 466, 499
 - definition of ... 438
 - foo ... 157
 - for ... 191, 194, 226-227, 234, 239, 246, 248, 269, 275, 296, 547, 683
 - definition of ... 226
 - Loop ... 226
 - foreground process group
 - definition of ... 35
 - foreground process
 - definition of ... 35
 - foreground ... 27, 35-36
 - foreground ... 27, 35, 240
 - fork()* ... 41, 139-140, 250, 778-779, 782, 784, 808
 - <form-feed>
 - definition of ... 36
 - fort77
 - FORTRAN compiler ... 817, 846
 - ... 813, 817-823
 - definition of ... 817
 - FORTRAN Development and Runtime Utilities
 - Options ... 813, 846
 - FORTRAN-66 ... 822
 - FORTRAN-8X ... 822
 - FORTRAN ... 12-14, 48, 179, 706, 725, 748, 777, 813, 816-819, 821-822, 843, 846
 - FORT_DEV ... 179, 777, 811
 - FORT_RUN ... 179, 777, 811
 - fpathconf()* ... 722-723, 811
 - definition of ... 811
 - fread()* ... 778
 - FS
 - awk variable ... 272
 - fstat()* ... 32, 35
 - FTAM ... 847
 - FTP ... 847
 - FUNC_NAME ... 282-283, 286-287
 - FUNC_NAME ... 290
 - Function Definition Command ... 230
 - Functions ... 277
 - fwrite()* ... 778
- ## G
- gawk ... 297
 - General Terms ... 26
 - General ... 1, 829
 - Generate Pathnames Matching a Pattern
 - ... 722, 844

Get Numeric-Valued Configurable Variables
... 722

Get POSIX Configurable Variables ... 722,
844

Get String-Valued Configurable Variables
... 722

getconf
— Get configuration values ... 442, 840
... 52, 107, 174-176, 179-180, 356-358,
418, 442, 444-446, 537, 730
definition of ... 442

getenv() ... 720, 786
definition of ... 786

getgrgid() ... 432

getgrnam() ... 431

GETLINE ... 282, 287

getlogin() ... 494

getopt ... 450

getopt() ... 153, 155, 257, 448, 451-452, 542,
601, 721, 796-797, 799
definition of ... 796

getopts
— Parse utility options ... 447, 840
... 51-52, 107, 194, 255, 447-452, 797,
799
definition of ... 447

getpwnam() ... 197, 433

getpwuid() ... 432

gid_t ... 36, 45

Global Command ... 409

Global Non-Matched Command ... 414

glob() ... 722, 799-803, 807
definition of ... 799
Error Return Values ... 802
flags Argument ... 800

globfree() ... 722, 799-801

<glob.h> ... 799-801

GLOB ... 773

GLOB_ABORTED ... 801

GLOB_APPEND ... 800-803

GLOB_DOOFFS ... 800-803

GLOB_ERR ... 800-801

GLOB_MARK ... 800

GLOB_NOCHECK ... 800-802

GLOB_NOESCAPE ... 800

GLOB_NOMATCH ... 801

GLOB_NOSORT ... 800

GLOB_NOSPACE ... 801

glob_t
definition of ... 800

GMT0 ... 373, 378

GNU make ... 712

GNU ... 297, 708-709, 712, 714-715, 749

Grammar Conventions ... 22

Grammar Rules ... 756

GRAVE ... 100-101

GREATAND ... 235, 238

grep
— File pattern searcher ... 452, 840
... 5, 47, 125, 127, 158, 453-458, 471,
709, 791, 834
definition of ... 452

group ID
definition of ... 36
effective ... 31
real ... 42
saved set- ... 42
supplementary ... 44

Grouping Commands ... 226

groups ... 465-466

groups
multiple (see supplementary group ID)

H

hard link
definition of ... 36

hd ... 536

head
— Copy the first part of files ... 459, 840
... 160, 459-462, 627
definition of ... 459

Header file ... 752

Headers and Function Prototypes ... 774

Help Command ... 410

Help-Mode Command ... 410

Here-Document ... 211

HEX_CHAR ... 94

hexdump ... 536

hierarchy
file ... 33

HIGH ... 77, 80, 114

HISTFILE
variable ... 195

HISTSIZE
variable ... 195

- home directory
 - definition of ... 36
- HOME**
 - variable ... 20, 103, 194, 197, 199, 252, 260, 322-323, 325, 403-404, 511, 526-527, 599
- \$HOME/nohup.out ... 528-529
- HOME ... 20, 136, 202-203, 252-253, 260, 528-529
- HP-UX ... 677
- HUPCL ... 614
- HUP ... 258
- I**
- IBM ... 58, 386-387
- ICANON ... 616
- ICRNL ... 615
- id
 - Return user identity ... 462, 840
 - ... 139, 462-463, 465-466
 - definition of ... 462
- IDENTIFIER ... 759-760
- IEEE P1003.2 ... 719, 829
- IEEE P1003.2a ... 685
- IEEE P1003.3.2 ... 826
- IEEE P1003.3 ... 826
- IEEE Std 100 ... 826
- IEEE Std 754 ... 535
- IEEE ... 7-8, 11, 48, 67, 346, 685, 776, 826, 829
- IEXTEN ... 616
- if ... 227-229, 247, 255, 269, 296
 - Conditional Construct ... 228
 - definition of ... 228
- IFS**
 - variable ... 107, 192-196, 203, 207-208, 357, 427, 577, 599-601
- IFS ... 194, 218, 357
- IGNBRK ... 615
- IGNCR ... 615
- IGNORE ... 96, 101
- IGNORE ... 77-80, 83, 696, 700-701, 714
- IGNPAR ... 615
- III ... 9, 378
- Implementation Conformance ... 12
- implementation defined ... 13, 15, 23, 26, 28, 32-33, 38-39, 41, 55, 57, 61, 63-65, 67, 69-70, 91, 103-107, 141, 144, 155, 209, 257, 272-273, 280, 288, 297, 300, 322-323, 325, 329, 332, 336, 360-362, 366-367, 395, 398-399, 402, 411, 431, 476-477, 481, 485-487, 490, 500-502, 504-506, 509-510, 521, 533, 549, 551, 553-558, 560, 592, 613, 616, 632, 641, 647, 661-663, 665, 699-701, 703, 711, 727-728, 731, 733-734, 738-740, 742, 753, 764, 769, 814, 818-819, 821, 833, 835-836
- definition of ... 23
- implementation
 - definition of ... 23
- in ... 191, 227, 270
- INCLUDE ... 714, 822
- incomplete line
 - definition of ... 36
- INCR_DECR ... 303, 305
- INCR ... 282, 286-287
- INCR_DECR ... 306
- INCR ... 290
- Inference Rules ... 703
- INITIAL ... 745
- INLCR ... 615
- INPCK ... 615
- Input Grammar ... 758
- Input Language ... 753
- Input Modes ... 615
- input() ... 746-747, 749
- Input/Output and General Functions ... 280
- Insert Command ... 410
- Interactive Global Command ... 410
- Interactive Global Not-Matched Command ... 414
- interactive shell
 - definition of ... 183
- Interface to the Lexical Analyzer ... 763
- Internal Macros ... 705
- Internationalization Proposal Areas ... 131
- Internationalization Requirements ... 129
- Internationalization Syntax ... 133
- Internationalization Technical Background ... 129
- interval expression ... 116, 119
- INTR ... 615-617
- INT ... 258, 260
- invoke
 - definition of ... 36
- IO_NUMBER ... 235, 238
- IO_NUMBER ... 233

IRV ... 60, 533, 535, 538
 IS1 ... 68, 74
 IS2 ... 68, 74
 IS3 ... 68, 74
 IS4 ... 68, 74
isatty() ... 654
 ISIG ... 616
 ISO 10646 ... 884-885
 ISO 1539 ... 12, 817, 821
 ISO 2022 ... 825
 ISO 2047 ... 825, 885
 ISO 3166 ... 825, 849
 ISO 4217 ... 12, 83
 ISO 4873 ... 12, 59, 885
 ISO 639 ... 825, 849
 ISO 6429 ... 825, 850, 885
 ISO 646 ... 884-885
 ISO 6937-2 ... 825, 885
 ISO 6937 ... 82
 ISO 7- ... 12
 ISO 8802-3 ... 341, 825
 ISO 8806 ... 825
 ISO 8859-1 ... 12, 58, 131, 849-850
 ISO 8859-2 ... 12, 58
 ISO 8859 ... 58-59, 825, 850, 885
 ISO 8- ... 12
 ISO 9999-1 ... 19
 ISO_10646
 Charmap ... 887
 ISO_10646 ... 850, 887
 ISO_8859-1
 Charmap ... 912
 ISO_8859 ... 887, 912
 ISO/AFNOR ... 48, 826
 ISO/IEC 10367 ... 825
 ISO/IEC 10646 ... 59, 825, 850
 ISO/IEC 646 ... 12, 30, 38, 48-49, 58, 60, 67,
 533, 535, 538, 572, 621, 850
 ISO/IEC 9899 ... 12, 50
 ISO/IEC 9945-1 ... 12, 50, 774
 ISO/IEC Conforming POSIX.2 Application
 ... 15
 IS1 ... 56
 IS2 ... 56
 IS3 ... 56
 IS4 ... 56

isspace() ... 677
 ISTRIP ... 615
 IXOFF ... 616
 IXON ... 616
 IX.1991 ... 378

J

JCL ... 385
 JIS ... 826, 886
 job control ... 36, 43, 52, 182, 258, 472, 476,
 602, 617, 622, 673, 837
 definition of ... 36
 Join Command ... 411
 join
 — Relational database operator ... 466,
 840
 ... 5, 177, 466-470
 definition of ... 466
 {JOIN_LINE_MAX} ... 177
 JTC1 ... 826

K

kill
 — Terminate or signal processes ... 471,
 840
 ... 51-52, 139, 259, 471, 473-476, 671-
 672, 836
 definition of ... 471
kill() ... 471-472
 KILL ... 474, 616-618, 621
 kl_DK
 — (Example) Greenlandic LC_TIME and LC_-
 MESSAGES ... 883
 KornShell ... 9, 185, 187, 190-191, 193, 195,
 198-199, 201-202, 204-208, 213, 215, 221,
 228, 231-232, 243, 245, 249-250, 254, 256-
 257, 260, 325, 357, 422, 451, 475, 529, 601-
 602, 637, 639, 673, 682, 826
 ksh ... 215, 257

L

LALR ... 761, 764-766, 827
 L_ANCHOR ... 122-123
LANG
 variable ... 103, 105-106, 161, 194, 265-
 266, 298, 302, 319, 323, 327, 330, 338,
 342, 348, 351, 355, 363, 370, 375, 382,

- 389, 396, 399-400, 403, 420, 423-424, 434, 440, 444, 449, 455, 460, 463, 468, 473, 478, 482, 485, 488, 491, 494, 497, 504, 511, 516, 519, 523, 527, 532, 540, 544, 554, 564-565, 568-569, 574, 577, 581, 585, 589, 600, 603, 608, 619, 625, 629, 634, 642, 646, 655, 658, 663, 666, 670, 675, 680, 689, 697, 717, 729, 737, 751, 815, 819
- LANG ... 484-485
- language binding ... 1, 9, 13-16, 26-28, 36-37, 44, 87, 174, 176, 179, 209, 719-723, 725, 771, 775, 786, 811, 845
- Language-Independent System Services ... 719, 844
- LC_COLLATE locale category ... 74, 76, 99, 101
- LC_COLLATE ... 74, 76, 95-96, 99, 101
- LC_CTYPE locale category ... 68, 99, 122
- LC_CTYPE ... 68, 94-95, 99, 122
- LC_MESSAGES locale category ... 92, 102
- LC_MESSAGES ... 92, 96-97, 102
- LC_MONETARY locale category ... 84, 101
- LC_MONETARY ... 84, 97, 101
- LC_NUMERIC locale category ... 88, 101-102
- LC_NUMERIC ... 88, 97-98, 101-102
- LC_*
 - definition of ... 50
- LC_*
 - variable ... 105, 144, 194, 482, 485, 488
- LC_ALL locale category ... 106, 485, 752, 767
- LC_ALL**
 - variable ... 103, 105, 161, 194, 265-266, 298, 302, 319, 323, 327, 330, 338, 342, 348, 351, 355, 363, 370, 375, 382, 389, 396, 399-400, 403, 420, 423-424, 434, 440, 444, 449, 455, 460, 463, 468, 473, 478, 482-483, 488, 491, 494, 497, 504, 511, 516, 519, 523, 527, 532, 540, 544, 554, 564-565, 568-569, 574, 577, 581, 585, 589, 600, 603, 608, 619, 625, 629, 634, 642, 646, 655, 658, 663, 666, 670, 675, 680, 689, 697, 717, 729, 737, 751, 815, 819
- LC_ALL ... 106, 483, 485, 752, 767
- LC_COLLATE Category Definition in the POSIX Locale ... 74
- LC_COLLATE locale category ... 29, 62, 65, 73, 77, 81-82, 130, 132, 173-174, 481, 485-486, 489, 647-649, 767
- LC_COLLATE ... 72
- LC_COLLATE**
 - variable ... 61, 103, 105-106, 195, 266, 351, 363, 403, 424, 434, 456, 469, 488, 504, 523, 554, 581, 589, 600, 608, 646, 737
- LC_COLLATE ... 29, 62, 65, 72-74, 77, 81-82, 130, 132, 173-174, 481, 484-487, 489, 647-649, 767, 850
- LC_CTYPE Category Definition in the POSIX Locale ... 68
- LC_CTYPE locale category ... 27, 29, 40, 46, 62, 65, 67, 71, 114, 131, 133-134, 279, 381, 481, 485-486, 489, 533, 539, 606, 648-649, 767
- LC_CTYPE ... 67
- LC_CTYPE**
 - variable ... 49, 61, 103, 105-106, 195, 266, 298, 302, 319, 324, 327, 330, 338, 342, 348, 351, 355, 363, 370, 375, 382, 389, 396, 402, 404, 420, 424, 434, 440, 444, 449, 456, 460, 464, 469, 473, 478, 482, 488, 492, 497, 504, 511, 516, 519, 524, 527, 532, 540, 544, 554, 565, 569, 577, 581, 585, 589, 600, 604, 608, 619, 625, 629, 634, 643, 646, 655, 658, 663, 667, 670, 675, 680, 689, 697, 717, 729, 737, 751, 815, 820
- LC_CTYPE ... 27, 29, 40, 46, 62, 65, 67-68, 71, 114, 131, 133-134, 279, 381, 481, 485-487, 489, 533, 539, 606, 648-649, 767
- LC_MESSAGES Category Definition in the POSIX Locale ... 92
- LC_MESSAGES locale category ... 26, 37, 63, 92, 363, 434, 464, 523-524, 554, 581, 812
- LC_MESSAGES ... 92
- LC_MESSAGES**
 - variable ... 61, 92, 104-106, 163, 195, 266, 298, 302, 319, 324, 327, 330, 338, 342, 348, 352, 355, 363, 370, 375, 383, 389, 396, 400, 404, 420, 424, 434, 440, 444, 449, 456, 460, 464, 469, 473, 478, 482, 488, 492-493, 495, 498, 504, 511, 516, 519, 524, 528, 532, 540, 544, 555, 565, 569, 575, 577, 581, 586, 589, 600, 604, 608, 619, 625, 629, 634, 643, 646, 655, 658, 663, 667, 670, 675, 680, 689, 697, 717, 729, 737, 751, 815, 820
- LC_MESSAGES ... 26, 37, 63, 92, 363, 434, 464, 485, 487, 523-524, 554, 581, 812, 850, 882-883

- LC_MONETARY Category Definition in the
POSIX Locale ... 84
- LC_MONETARY locale category ... 62, 83-84,
86
- LC_MONETARY ... 83
- LC_MONETARY**
variable ... 61, 104-106
- LC_MONETARY ... 62, 83-84, 86, 88, 485, 487
- LC_NUMERIC Category Definition in the POSIX
Locale ... 88
- LC_NUMERIC locale category ... 63, 87-88,
170-171, 267, 296, 485
- LC_NUMERIC ... 87
- LC_NUMERIC**
variable ... 61, 104-105, 155, 266, 532,
569, 609
- LC_NUMERIC ... 63, 87-88, 170-171, 267,
296, 485, 487
- LC_TIME Category Definition in the POSIX
Locale ... 89
- LC_TIME locale category ... 63, 88, 374, 505,
566
- LC_TIME ... 88
- LC_TIME**
variable ... 61, 104-106, 376, 389, 504,
555, 565, 690-691
- LC_TIME ... 63, 88-89, 91, 374, 378, 485, 487,
505, 566, 850, 882-883
- LC_TYPE ... 850
- LC_TIME locale category ... 89, 102
- LC_TIME ... 89, 98-99, 102
- LCURL ... 759-760
- LDBL_MANT_DIG ... 537
- LDFLAGS ... 706-707
- leap seconds ... 377
- LEFT ... 759-760, 768
- LESSAND ... 235, 238
- LESSGREAT ... 235, 238
- LETTER ... 303-305
- LETTER ... 306, 313
- lex
 - Generate programs for lexical tasks
... 736, 845
... 4, 25, 41, 44, 129, 137, 163, 165,
172, 730, 734, 736-742, 744-749, 753,
766, 822
 - Actions ... 744
 - definition of ... 736
 - Definitions ... 740
 - ERE Precedence ... 743
 - Escape Sequences ... 743
 - Regular Expressions ... 741
 - Rules ... 741
 - Table Size Declarations ... 741
 - User Subroutines ... 741
- Lexical Structure of the Grammar ... 753
- LEX ... 162, 706-707
- lex.yy.c ... 165, 736, 738-739, 741, 744,
749
- LFLAGS ... 706-707
- /lib ... 109
- libc.a ... 727, 731
- LIBDIR ... 198
- libf.a ... 818, 821
- libl.a ... 727, 731, 749
- libm.a ... 727, 731
- Libraries ... 704
- liby.a ... 727, 731
- LIGATURE ... 100-101
- {LIMIT} ... 173
- Limits ... 764
- <limits.h> ... 20, 774-775
- Line Number Command ... 415
- line ... 835
- line
 - definition of ... 36
- LINENO**
variable ... 195
- {LINE_MAX} ... 45, 47, 64, 160-161, 175, 177-
179, 185, 265, 277, 371, 417-418, 439, 460,
466, 510, 512, 542, 596, 623, 668, 678-679,
682, 776
- {LINE_MAX} ... 19
- LINE_MAX ... 173-174
- {LINE_MAX} ... 174
- LINE_MAX ... 775
- {LINE_MAX} ... 775, 811
- link (see directory entry)
- link count
 - definition of ... 37
- link
 - definition of ... 37
- link() ... 3, 36, 477, 479
- {LINK_MAX} ... 178
- {LINK_MAX} ... 723
- List Command ... 411
- Lists ... 222
- ln
 - Link files ... 476, 840
... 3, 36, 47, 366, 476-480
 - definition of ... 476

- Local Modes ... 616
- /local ... 109
- local ... 231
- /local/bin ... 155
- Locale Control ... 723, 844
- locale definition file ... 63
- Locale Definition Grammar ... 93
- Locale Definition ... 63
- Locale Grammar ... 94
- Locale Lexical Conventions ... 93
- Locale String Definition Guideline ... 849
- locale
 - Get locale-specific information ... 480, 840
- Locale ... 61
- locale ... 480-482, 484-485
- Locale ... 830
- locale
 - definition of ... 37
- locale
 - definition of ... 480
- localeconv()* ... 86
- localedef
 - Define locale environment ... 486, 840
 - ... 20, 44, 58-63, 65-66, 91, 105, 378, 483-484, 486-490
 - definition of ... 486
- LOCALEDEF ... 179, 777, 811
- <locale.h> ... 774, 812
- locale ... 5, 14, 20, 26-27, 29, 37, 40, 44, 46, 48-49, 55, 58-68, 71-74, 81, 83-84, 86-99, 103-106, 110, 114, 122, 130-133, 143-144, 153, 155, 161, 170-171, 173-174, 179-180, 215, 245, 254, 265-267, 271, 279, 296, 298, 302, 309, 316-317, 319, 323-324, 327, 330, 338, 342, 348, 350-352, 355, 363, 370, 373-375, 377-378, 382, 389, 395-396, 399-400, 403-404, 420, 423-424, 434, 440, 444, 449, 454-456, 460, 463-464, 468-469, 473, 478, 480-492, 494, 497, 499, 502, 504-505, 510-511, 516, 519, 523-524, 527, 532-5 33, 535-538, 540, 544, 554, 564-569, 572, 574, 577, 581, 585, 589, 600, 603-604, 606-609, 619-620, 625, 629, 634, 642-643, 646-648, 655, 658, 663, 666-667, 670, 675, 680, 689, 691, 697, 717, 719, 723, 729, 737, 748, 751-752, 767, 774, 777, 788, 812, 815, 819-820, 830, 840, 844, 846-850, 887
- locate ... 438
- LOC_NAME ... 94
- logger
 - Log messages ... 491, 840
 - ... 491-493, 513
 - definition of ... 491
- login name
 - definition of ... 37
- login session
 - definition of ... 48
- login
 - definition of ... 37
- logname
 - Return user's login name ... 494, 840
 - ... 494-496
 - definition of ... 494
- LOGNAME**
 - variable ... 104, 108, 197, 496
- logout ... 260
- {LONG_MAX} ... 145, 150
- {LONG_MIN} ... 150
- LONG_NAME_OS ... 138
- LOWER-CASE ... 100-101
- LOWER ... 101
- LOW ... 80
- LOW_VALUE ... 100-101
- lp
 - Send files to a printer ... 496, 841
 - ... 107, 178, 441, 493, 496-497, 499-501, 513, 816
 - definition of ... 496
- LPDEST**
 - variable ... 107, 496, 498-501
- lpr ... 501
- {LP_LINE_MAX} ... 178
- ls
 - List directory contents ... 502, 841
 - ... 81, 107, 138, 142, 150, 202, 221, 332, 336, 437, 502-504, 507-509, 555, 558-559, 681, 691, 694
 - definition of ... 502
- lseek()* ... 32, 630, 735, 823
- M**
- macro
 - feature test ... 32
- MACRO ... 715
- Macros ... 702

- macros
 - feature test ... 772
- Mail ... 514
- MAILRC**
 - variable ... 107, 511, 514
- MAIL**
 - variable ... 107
- mailto ... 513
- mailx
 - Process messages ... 510, 841
 - ... 107, 143, 493, 510-514, 835
 - definition of ... 510
- main()* ... 745-746, 751, 753, 763-764, 766, 792, 796, 803
- make ... 4, 25, 44, 142, 155, 160, 162, 185, 198, 232, 692, 694-716, 767
 - definition of ... 695
 - GNU version ... 712
- Makefile Execution ... 700
- Makefile Syntax ... 699
 - ./Makefile ... 699
 - ./makefile ... 699
 - ./Makefile ... 700
 - ./makefile ... 700
 - ./Makefile ... 711
 - ./makefile ... 711
- MAKEFLAGS**
 - variable ... 107, 696, 698, 700, 710, 712, 714
- MAKEFLAGS ... 696, 698
- MAKE ... 707-708
- {MAKE} ... 708
- MAKE ... 710, 712
- {MAKE} ... 712
- MAKESHELL**
 - variable ... 713
- malloc()* ... 791, 803, 810
- many-to-many substitution ... 73
- Mark Command ... 411
- MARK ... 759
- matched
 - definition of ... 110
- Matching Expression ... 425
- matching list ... 113
- Mathematic Functions ... 145
- <math.h> ... 730
- MAX ... 773
- {MAX_CANON} ... 177-178
- {MAX_CANON} ... 723
- {MAX_INPUT} ... 178-179
- {MAX_INPUT} ... 723
- may
 - definition of ... 23
- mb_cur_max ... 56
- {MEMSIZE} ... 764
- message formats ... 92
- messaging ... 92
- META_CHAR ... 122, 124
- META_CHAR ... 121
- MIL-STD-1753 ... 822
- MIL-STD-1753 ... 822
- MIN ... 618
- Miscellaneous Conventions ... 22
- mkdir ... 515
- mkdir
 - Make directories ... 514, 841
 - ... 514-517, 520
 - definition of ... 514
- mkdir()* ... 32, 514, 517, 549
- mkfifo
 - Make FIFO special files ... 518, 841
 - ... 518-520, 835
 - definition of ... 518
- mkfifo()* ... 32, 518, 520
- mknod ... 366, 835
- mktemp ... 547
- MOD_ASSIGN ... 282, 286-287
- mode
 - definition of ... 37
- mode_t* ... 46
- Modified Field Descriptors ... 374
- MOD_ASSIGN ... 290
- monetary formatting ... 83
- more ... 213, 320
- Move Command ... 411
- MS/DOS ... 3
- MUL_ASSIGN ... 282, 286-287
- MUL_OP ... 303, 305
- MUL_ASSIGN ... 290
- MUL_OP ... 306
- multibyte character ... 28
- multicharacter collating element
 - definition of ... 37
- multicharacter collating elements ... 72

multiple groups (see supplementary group ID)
 multiple weights and equivalence classes
 ... 72

mv
 — Move files ... 521, 841
 ... 480, 521-526, 584
 definition of ... 521
 MVS/TSO ... 3
 /mybin ... 421
 mygrep ... 421

N

NAK ... 68, 74
 NAK ... 56, 618
 name
 definition of ... 183
 login ... 37
 user ... 45
 NAME ... 235-237, 282-287
 {NAME_MAX} ... 34, 39, 178, 446, 543, 561,
 692
 NAME_MAX ... 446
 {NAME_MAX} ... 723
 NAME ... 22, 233-234, 270, 290
 <National Body> Conforming POSIX.2 Applica-
 tion ... 16
 nawk ... 293
 negative response
 definition of ... 37
 <newline> ... 437
 definition of ... 37
 NEWLINE ... 235, 238, 282-283, 287, 303-304
 NEWLINE ... 22, 233, 288, 305, 312
 NEW ... 715
 next ... 275
 NF-1 ... 292
 NF
 awk variable ... 272
 {NGROUPS_MAX} ... 44, 139, 178, 445, 462
 NGROUPS_MAX ... 446
 {NGROUPS_MAX} ... 723
 Ninth Edition UNIX ... 231, 315, 402, 572
 NIX ... 158
 nl ... 835
 {NNONTERM} ... 765
 NO-ACCENT ... 100-101

NO-ACCENT ... 101
 noclobber option ... 210, 213-214, 255,
 546-547
 noexpr ... 37, 92, 102
 NOFLSH ... 617
 nohup
 — Invoke a utility immune to hangups
 ... 526, 841
 ... 51, 215, 357, 422, 526-529, 682
 definition of ... 526
 nohup.out ... 526-528
 ./nohup.out ... 529
 NO_MATCH ... 282, 285-287
 NONASSOC ... 759-760
 nonmatching list ... 113
 nonprintable ... 47, 135, 411, 417, 503, 533,
 556, 565, 592, 595, 606, 620
 Normative References ... 12, 829
 NO_MATCH ... 290
 NOTE ... 239
 NOTES ... 21
 NOT ... 222, 269, 433, 525, 583
 NPROC ... 712
 {NPROD} ... 765
 nroff ... 372, 833
 NR
 awk variable ... 272
 {NSTATES} ... 765
 {NTERMS} ... 765
 Null Command ... 415
 null string
 definition of ... 31, 37
 NULL ... 101
 NULL ... 781, 791, 802, 810
 NUL ... 68, 74
 NUL
 definition of ... 37
 NUL ... 37, 45, 47, 55, 78, 111, 113, 118, 242,
 274, 288, 418, 533, 651, 742, 745, 754, 764
 Number Command ... 411
 NUMBER ... 94, 97-98, 282, 285, 287, 303-304
 NUMBER ... 269, 288, 305-306, 759-760
 number-sign
 definition of ... 38
 numeric formatting ... 87

O

O_APPEND ... 141, 210, 630
 object file
 definition of ... 38
 obsolescent features ... 4-5, 15, 24-25, 104, 107, 135, 154, 198, 205, 227, 254, 256, 332, 386, 402-403, 419, 432, 452-454, 458-459, 461, 466-467, 470-472, 475, 537, 566-567, 605-606, 608, 610-611, 613, 623-624, 626, 640, 642, 644, 654-656, 658, 660, 665-666, 669, 736, 834
 obsolescent
 definition of ... 24
O_CREAT ... 361
OCTAL_CHAR ... 94
 od
 — Dump files in various formats ... 530, 841
 ... 530, 532-534, 536-537, 709
 definition of ... 530
 Named Characters ... 534
OFF ... 24, 46, 111, 185
off_t ... 786
OFMT
 awk variable ... 272
OFMT ... 271-272, 276, 295
OFS
 awk variable ... 272
OFS ... 271, 276, 292
OLDPWD
 variable ... 325
ONESHELL ... 714
 one-to-many mapping ... 72
O_NONBLOCK ... 143, 602
 Open File Descriptors for Reading and Writing.
 ... 212
 open file
 definition of ... 38
open() ... 32, 141, 143, 210, 360-361, 387, 602
opendir() ... 801
{OPEN_MAX} ... 178, 212
{OPEN_MAX} ... 723
 operand
 definition of ... 38
 operator
 definition of ... 183
OPOST ... 616
OPTARG
 variable ... 107, 447-448, 451

OPTARG ... 451
OPTERR
 variable ... 451
OPTIND
 variable ... 107, 447-449, 451
OPTIND ... 451
 option
 definition of ... 38
 Optional Facility Configuration Values
 ... 179
 option-argument
 definition of ... 38
ORD_CHAR ... 122-125
order_end
 Keyword ... 81
 ordering by weights ... 73
order_start
 Keyword ... 77
O_RDONLY ... 142
ORD_CHAR ... 121
OR_IF ... 235-236
 OR Lists ... 225
ORS
 awk variable ... 272
ORS ... 276
O_TRUNC ... 141, 360, 387
 Output Modes ... 616
 Output Statements ... 276
 Overall Program Structure ... 267
O_WRONLY ... 360-361

P

P.0 ... 610
PARALLEL ... 712
 Parameter Expansion ... 199
 parameter
 definition of ... 183
 Parameters and Variables ... 192, 831
PARENB ... 614
 parent directory
 definition of ... 38
 parent process ID
 definition of ... 38
 parent process
 definition of ... 38
PARMRK ... 615

- PARODD ... 614
- passwd ... 496
- paste
 - Merge corresponding or subsequent lines of files ... 538, 841
 - ... 178, 371, 538, 540-542
 - definition of ... 538
- patch ... 394
- path prefix
 - definition of ... 39
- pathchk
 - Check pathnames ... 543, 841
 - ... 213, 543-547
 - definition of ... 543
- pathconf()* ... 39, 175, 443, 722-723, 811
 - definition of ... 811
- pathname component
 - definition of ... 39
- Pathname Expansion ... 208
- pathname resolution ... 26
- Pathname Resolution ... 143
- pathname resolution
 - definition of ... 39
- pathname
 - absolute ... 26
 - definition of ... 39
 - relative ... 42
- pathname ... 19-20, 26-27, 30, 34-35, 38-40, 42, 46, 48, 61, 103-105, 107-108, 139, 142-143, 155, 157, 181, 194-197, 199, 204-205, 208-209, 213, 219-220, 234, 243-245, 255, 264, 272, 276, 280, 297, 299-301, 318, 320, 323-324, 326, 329, 338, 341-342, 347, 351, 359, 363-364, 369, 380, 388, 390-391, 395, 398, 403, 408-409, 412, 414-415, 427, 430-433, 435-436, 439, 442-443, 454-455, 460-461, 468, 470, 476-477, 487, 497, 502-505, 511, 515, 518, 521, 523-524, 527, 531, 539, 543-548, 550, 552-556, 558, 562, 564, 574-575, 579-582, 585, 588, 598-599, 608, 624, 628, 630, 641-642, 644, 655-656, 666, 674, 676, 687, 689, 696, 699, 703, 711, 716, 719, 722, 726-729, 737, 751, 778, 782, 785, 794-795, 799-804, 807, 814, 817-818, 820, 838-839, 841, 844-845
- PATH**
 - variable ... 51-53, 104, 107-108, 162, 195, 198, 219-220, 248-249, 252, 266, 292, 325, 354-358, 420-421, 434, 445, 528, 598, 600, 679, 781, 809-810
- {PATH_MAX} ... 39, 175, 178, 430, 543, 584
- PATH_MAX ... 446
- {PATH_MAX} ... 723
- PATH ... 19, 195, 198, 221, 253, 292, 357, 421, 443
- pattern matching notation ... 72, 79, 195, 200, 208, 227
- Pattern Matching Notation ... 242
- pattern matching notation ... 242, 244-245, 432, 434, 436-437, 553-554, 560, 600, 719, 721, 793, 799
- Pattern Matching Notation ... 832
- Pattern Matching ... 721, 844
- Pattern Ranges ... 275
- pattern
 - definition of ... 40
- Patterns Matching a Single Character ... 242
- Patterns Matching Multiple Characters ... 244
- Patterns Used for Filename Expansion ... 244
- Patterns ... 274
- pax
 - Portable archive interchange ... 548, 841
 - ... 243, 367, 538, 545, 548-561, 692, 709, 795, 833
 - definition of ... 548
- pclose()* ... 276, 280, 720, 782-785
 - definition of ... 782
- PDT ... 377, 393
- PECULIAR ... 100-101
- Perform Word Expansions ... 722, 844
- period
 - definition of ... 40
- Periods in BREs ... 113
- Periods in EREs ... 118
- permission
 - file ... 34
- permissions
 - definition of ... 40
 - file access ... 32, 40
- pid_t* ... 41
- Pipe Communications with Programs ... 720
- pipe ... 34
 - definition of ... 40
- pipe()* ... 40, 784
- Pipelines ... 221
- pipe ... 6, 34, 40, 106, 143, 160, 172, 178, 192-193, 216, 221-222, 236, 241, 255, 266, 272, 276, 280, 293, 317, 356, 395, 437, 499, 529, 558, 624, 627, 630-631, 681, 720, 782-

- 785, 816
- {PIPE_BUF} ... 178
- {PIPE_BUF} ... 723
- PL/1 ... 314
- popen()* ... 2, 8, 51, 53, 181, 276, 280, 720, 778, 782-784, 811
 - definition of ... 782
- portable character set ... 40, 54-55, 58-59, 62, 183, 264, 289, 405, 411, 490, 701-702
 - definition of ... 40
- portable filename character set
 - definition of ... 40
- portable filenames ... 34
- positional parameter
 - definition of ... 183
- Positional Parameters ... 192
- POSIX Locale ... 68, 74, 84, 88-89, 92
- posixconf ... 446
- posixconf()* ... 810
- posixlog ... 493
- POSIX Locale ... 27, 46, 61-63, 66, 91, 105, 115, 161, 296, 309, 348-349, 374, 376-377, 383, 390-391, 435, 454, 464, 481, 485, 488, 505-507, 510, 535, 565-567, 618, 620-621, 655, 676, 691, 737, 748, 767, 848-849
- POSIX Symbols ... 772
- POSIX.1 C Numerical Limits ... 777
- POSIX.1 Numeric-Valued Configurable Variables ... 723
- POSIX.1
 - definition of ... 50
- POSIX.1 ... 1-4, 7, 9-11, 13-14, 16-17, 20, 24-46, 48-53, 63, 66, 92, 103, 105-107, 138-144, 156, 160, 162, 175-178, 194, 197, 209-210, 212, 215, 219-220, 240-241, 246, 258, 271, 300, 326, 328, 334, 336-337, 339-340, 343, 349, 358-361, 367, 388, 395, 398, 422, 430, 432-433, 436-437, 442-443, 462, 471-472, 474-477, 494, 502-503, 505, 509, 514, 517-518, 520-521, 525, 537, 543, 545, 547, 549, 552-556, 558-560, 580, 602, 613-617, 621-622, 626, 639-640, 644, 654-655, 661-662, 665, 673, 678, 683, 719-720, 722-723, 726-727, 730, 734, 771-772, 774-780, 783-786, 801-802, 809-813, 817-818, 829, 835
- POSIX.2 Reserved Header Symbols ... 773
- POSIX.2
 - abbreviation ... 50
 - definition of ... 50
- {_POSIX2_C_BIND} ... 14, 180
- _POSIX2_C_BIND ... 730
- {_POSIX2_C_DEV} ... 14, 180
- {_POSIX2_C_DEV} ... 811
- {_POSIX2_FORT_DEV} ... 14, 180
- {_POSIX2_FORT_RUN} ... 14
- {_POSIX2_LINE_MAX} ... 776
- {_POSIX2_LOCALEDEF} ... 14, 61, 66, 486
- POSIX.2 ... 2-3, 6-11, 13-16, 23-24, 41, 46-50, 60, 62, 71-72, 81, 83, 104, 106-107, 109, 113-115, 127-128, 136-140, 142-145, 150, 154, 162, 166, 168
- POSIX2 ... 173-174
- POSIX.2 ... 175-177
- POSIX2 ... 179-180
- POSIX.2 ... 182, 185, 187-188, 190, 196, 198, 205, 207, 212, 215-216, 220-222, 231-232, 245, 247, 249, 253, 259, 261, 263, 294-296, 316, 335-336, 340, 343, 349, 356-358, 365-366, 377, 387, 416-418, 426, 436-437, 443, 445
- POSIX2 ... 446
- POSIX.2 ... 458, 462, 474-475, 493, 500-501, 508, 510, 512-514, 535, 537, 542, 550-551, 558, 560-561, 584, 587, 596, 601-602, 613, 621, 627, 635, 637-639, 650-651, 653, 665, 672, 681-683, 692-693, 708-709, 712-715, 721-723
- POSIX2 ... 730
- POSIX.2 ... 732-734, 748-749, 769, 771-774
- POSIX2 ... 774
- POSIX.2 ... 775
- POSIX2 ... 775
- POSIX.2 ... 776
- POSIX2 ... 776
- POSIX.2 ... 777
- POSIX2 ... 777
- POSIX.2 ... 779, 782-784, 792, 810-811
- POSIX2 ... 811
- POSIX.2 ... 813, 816, 822, 826, 847
- {_POSIX2_SW_DEV} ... 14, 180
- _POSIX2_VERSION ... 774
- {_POSIX2_VERSION} ... 776
- {_POSIX2_VERSION} ... 811
- POSIX.3 ... 2, 11, 161
- POSIX.5 ... 9
- POSIX.6 ... 325, 508-509, 837
- POSIX.7 ... 4, 7, 493

- POSIX.9 ... 813, 822
 - {_POSIX_C_DEV} ... 811
 - POSIX_CHOWN_RESTRICTED ... 723
 - _POSIX_C_SOURCE ... 772-773
 - POSIX_C_SOURCE ... 772-774
 - {_POSIX_JOB_CONTROL} ... 36
 - POSIX_JOB_CONTROL ... 723
 - {_POSIX_LOCALEDEF} ... 180
 - {_POSIX_NAME_MAX} ... 543
 - {_POSIX_NO_TRUNC} ... 39, 547
 - POSIX_NO_TRUNC ... 723
 - {_POSIX_PATH_MAX} ... 543
 - {_POSIX_SAVED_IDS} ... 139
 - POSIX_SAVED_IDS ... 723
 - POSIX_SOURCE ... 772, 777
 - {_POSIX_VDISABLE} ... 617, 621
 - POSIX_VDISABLE ... 19, 723
 - _POSIX_VERSION ... 730
 - {_POSIX_VERSION} ... 811
 - POSIX_VERSION ... 730, 774, 776
 - POW_ASSIGN ... 282, 286-287
 - POW_ASSIGN ... 290
 - PPID**
 - variable ... 195
 - pr
 - Print files ... 562, 841
 - ... 321, 499, 562-567, 835
 - definition of ... 562
 - PRECIOUS ... 698, 702, 714
 - PREC ... 759-760
 - Preserve Historical Applications ... 4
 - Preserve Historical Implementations ... 6
 - Print Command ... 412
 - print ... 272
 - printable character
 - definition of ... 40
 - PRINTER**
 - variable ... 107, 496, 498, 500-501
 - printf
 - Write formatted output ... 568, 841
 - ... 5, 8, 155, 271-272, 279, 401-402, 509, 541, 568, 570-573
 - definition of ... 568
 - printf()* ... 155, 160, 163, 165, 168, 171-172, 536-537, 573, 677, 734
 - privileges (see appropriate privileges)
 - Process Attributes ... 138
 - process group ID ... 41, 138-139, 474
 - definition of ... 41
 - process group leader
 - definition of ... 41
 - process group
 - background ... 27
 - definition of ... 41
 - foreground ... 35
 - leader ... 41
 - process ID ... 38, 41, 138-139, 164, 193, 224, 240, 474-475, 669-673, 783
 - definition of ... 41
 - parent ... 38
 - process
 - background ... 26
 - definition of ... 41
 - foreground ... 35
 - parent ... 38
 - PROCESSING ... 546-547
 - PROCLANG**
 - variable ... 108
 - program
 - definition of ... 41
 - Programs Section ... 758
 - Prompt Command ... 412
 - PS1**
 - variable ... 107, 195, 602
 - PS2**
 - variable ... 107, 195, 602
 - PS4**
 - variable ... 195
 - PWB ... 638
 - pwd
 - Return working directory name ... 574, 841
 - ... 574-576
 - definition of ... 574
 - PWD**
 - variable ... 195, 252, 325
 - PWD ... 252-253
- ## Q
- q ... 418
 - Quit Command ... 412
 - Quit Without Checking Command ... 412
 - QUIT ... 258, 260, 616-617
 - Quote Removal ... 209
 - QUOTED_CHAR ... 122-124

QUOTED_CHAR ... 121

Quoting ... 185, 831

R

R_ANCHOR ... 122-123

range expression ... 114

ranlib ... 694

RAW ... 622

RCS ... 712

RCURL ... 759-760

Read Command ... 412

read

— Read a line from standard input

... 576, 841

... 51-52, 160, 194, 196, 255, 576-579,
599, 835

definition of ... 576

read() ... 32, 142-143, 383*readdir()* ... 801

read-only file system

definition of ... 41

readonly

— Set read-only attribute for variables

... 253

... 253

definition of ... 253

real group ID ... 36, 42, 138-139, 464, 597

definition of ... 42

real user ID ... 42, 45, 138-139, 464, 597

definition of ... 42

Redirecting Input ... 210

Redirecting Output ... 210

redirection operator

definition of ... 184

Redirection ... 209, 831

redirection

definition of ... 184

regcomp() ... 2, 127, 129, 721, 786-790, 792,
795*cflags* Argument ... 787

definition of ... 786

regexec() Return Values ... 790*regerror()* ... 786, 789, 791, 793

definition of ... 786

regexec() ... 721, 786-790, 792-793, 795

definition of ... 786

eflags Argument ... 787

<regex.h> ... 786-787, 789-790, 793

regex_t

definition of ... 787

regfree() ... 721, 786, 789

definition of ... 786

regmatch() ... 791*regmatch_t*

definition of ... 787

regoff_t ... 786, 792

definition of ... 786

REG ... 773

REG_BADBR ... 789

REG_BADPAT ... 790

REG_BADRPT ... 789

REG_EBRACE ... 790, 793

REG_EBRACK ... 790

REG_ECOLLATE ... 790

REG_ECTYPE ... 790

REG_EESCAPE ... 790

REG_EPAREN ... 790

REG_ERANGE ... 790

REG_ESPACE ... 790

REG_ESUBREG ... 790

REG_EXTENDED ... 786-787, 791

REG_FILENAME ... 793, 795

REG_FSLASH ... 795

REG_ICASE ... 786, 791

REG_NEWLINE ... 786, 789

REG_NOMATCH ... 789

REG_NOSUB ... 786-788, 791-792

REG_NOTBOL ... 786, 789-790, 792

REG_NOTEOL ... 786, 789, 792

REG ... 790

regsub() ... 792

Regular Built-in Utilities ... 51

Regular Expression Definitions ... 110

Regular Expression General Requirements
... 111

Regular Expression Grammar ... 121

Regular Expression Matching ... 721, 844

Regular Expression Notation ... 110, 830

regular expression ... 2, 5-6, 11, 25-27, 32,
37, 40, 42, 50, 66, 72, 79-80, 83, 92-93, 103,
110-113, 117-118, 121-131, 134-135, 137,
173-174, 200, 242-243, 264, 266, 272-274,
278-279, 288, 291, 293-294, 297, 363, 403-
405, 417, 424-425, 427, 434, 453-456, 523-
524, 552, 554, 581, 589-591, 593, 595, 650-
651, 719, 721, 737, 739-744, 747-749, 786-
793, 830, 844-845

- definition of ... 42
- Regular Expressions ... 273
- regular file
 - definition of ... 42
- rejected utilities ... 832
- REJECT ... 744
- relative pathname
 - definition of ... 42
- REL_OP ... 303-304
- REL_OP ... 306
- rename()* ... 521, 525-526
- Required Files ... 109, 830
- Requirements ... 12
- RE and Bracket Expression Grammar ... 122
- RE Bracket Expression ... 113
- RE
 - abbreviation ... 50
- {RE_DUP_MAX} ... 116, 119, 121, 178
- RE_DUP_MAX ... 173-174
- {RE_DUP_MAX} ... 174
- RE_DUP_MAX ... 775
- {RE_DUP_MAX} ... 775, 811
- Reserved Words ... 190, 831
- return
 - Return from a function ... 254
 - ... 230, 254
 - definition of ... 254
- RIGHT ... 759-760
- RING-ABOVE ... 100
- RLENGTH
 - awk variable ... 272
- RLENGTH ... 278, 293
- rm
 - Remove directory entries ... 579, 841
 - ... 176, 479, 546, 579-584
 - definition of ... 579
- rmdir
 - Remove directories ... 584, 842
 - ... 517, 584-587
 - definition of ... 584
- rmdir()* ... 142, 580
- {RM_DEPTH_MAX} ... 178
- R_OK ... 798
- root directory ... 138
 - definition of ... 42
- rsh ... 602
- RS
 - awk variable ... 272

- RSTART
 - awk variable ... 273
- RSTART ... 278, 293
- RULES ... 707

S

- Sample National Profile ... 847
- Sample *pclose()* Implementation ... 785
- Sample *system()* Implementation ... 781
- saved set-group-ID ... 36, 138
 - definition of ... 42
- saved set-user-ID ... 42, 45, 138
 - definition of ... 42
- saved-set-group-ID ... 139
- saved-set-user-ID ... 139
- scanf()* ... 168, 172
- SCCS ... 436, 711-712, 833-834, 836-838
- SCCS/s.Makefile ... 711
- Scope of Danish National Locale ... 850
- Scope ... 1, 829
- SC22 ... 884
- SC_2 ... 811
- SC2 ... 826
- SC_BC_BASE_MAX ... 811
- SC_BC_DIM_MAX ... 811
- SC_BC_SCALE_MAX ... 811
- SC_BC_STRING_MAX ... 811
- SC_COLL_WEIGHTS_MAX ... 811
- SC_EXPR_NEST_MAX ... 811
- SC_LINE_MAX ... 811
- SC_POSIX_C_BIND ... 446
- SC_POSIX_C_DEV ... 811
- SC_RE_DUP_MAX ... 811
- SC_VERSION ... 811
- sdb ... 836
- seconds since the Epoch
 - definition of ... 42
- security considerations ... 32
- security controls
 - additional ... 32
 - alternate ... 32
 - extended ... 32
- sed
 - Stream editor ... 587, 842
 - ... 25, 41, 44, 160, 178, 194, 320, 466,
 - 537, 560, 587-588, 590-591, 594-596,
 - 627, 683, 833, 837
 - Addresses ... 590

- definition of ... 587
- Editing Commands ... 591
- Regular Expressions ... 590
- {SED_PATTERN_MAX} ... 178
- sendto ... 513
- Sequential Lists ... 225
- session leader
 - definition of ... 43
- session lifetime
 - definition of ... 43
- session membership ... 138
- session ... 27, 35, 43
 - definition of ... 43
- session ... 27, 35, 43, 48, 138-139, 174, 176, 216, 260
- set
 - Set/unset options and positional parameters ... 254
 - ... 192, 194, 210, 213, 240, 254-257, 260, 357, 547, 597, 601
 - definition of ... 254
- setbuf() ... 321
- setbuffer() ... 321
- setgid() ... 31, 42
- set-group-ID ... 138-139, 366
- set-group-ID-on-execution ... 34, 331-332
- <setjmp.h> ... 730
- setlocale() ... 62, 130-133, 143, 296, 812
- setpgid() ... 43
- setsid() ... 43
- setuid() ... 31, 42
- set-user-ID scripts ... 602
- set-user-ID ... 138-139, 336, 366, 506, 632
- set-user-ID-on-execution ... 34, 331-332
- setvbuf() ... 321
- SGML ... 8
- sh — Shell
 - the standard command language interpreter ... 597, 842
- sh ... 43, 107, 144, 160, 167, 181-182, 192-194, 215-216, 255, 257, 291, 597-602, 636, 639, 703, 778-779, 781-783, 802, 807-808
 - definition of ... 597
- shall
 - definition of ... 24
- Shell Command Interface ... 720, 844
- Shell Command Language ... 181, 831
- Shell Commands ... 216, 831
- Shell Definitions ... 183, 831
- Shell Escape Command ... 415
- shell execution environment ... 184, 193, 209, 217, 224
- Shell Execution Environment ... 240
- shell execution environment ... 240-241, 250, 322, 325, 448, 450, 576, 578, 657, 660, 669
- Shell Execution Environment ... 831
- Shell Grammar Lexical Conventions ... 233
- Shell Grammar Rules ... 233
- Shell Grammar ... 233, 831
- shell script
 - definition of ... 43
- shell
 - definition of ... 43
- shell ... 1-4, 6-8, 10, 13, 25, 28, 30, 36, 41, 43-47, 49, 51-53, 60, 105, 107, 109, 133, 145, 154, 165, 167, 176, 179, 181-196, 198-209, 211, 213-221, 223-224, 226, 228, 230-233, 236, 239-243, 245-246, 248-260, 291, 296, 300, 314, 320-322, 325, 354, 357, 409, 412, 415, 422, 426-427, 429, 436-437, 445, 447-448, 450-452, 466, 471, 475-476, 479-480, 482, 486, 513, 520, 529, 541, 546-547, 558, 573, 576-579, 583, 597-599, 601-602, 614, 621-622, 635-637, 639, 650, 653, 657, 660, 669, 671-673, 681, 683, 703, 709, 713-714, 719-722, 778, 781-782, 799, 802-808, 829, 831, 834-838, 842, 844-845
- SHELL**
 - variable ... 105, 107, 698, 703, 782
- SHELL ... 698, 703
- shift
 - Shift positional parameters ... 258
 - ... 256
 - definition of ... 258
- should
 - definition of ... 24
- SIGABRT ... 259, 472
- SIGALRM ... 259, 472, 604-605
- SIGCHLD ... 778-781
- SIGHUP ... 165, 259, 404, 472, 526, 528-529, 698, 713, 784
- SIGINT ... 165, 224, 240, 259, 383, 404, 410, 472, 628-629, 698, 713, 778-779, 781, 784
- SIGKILL ... 258-259, 472, 474
- signal
 - definition of ... 43
- <signal.h> ... 730

- Signals and Error Handling ... 240, 831
- SIGNULL ... 475
- SIGQUIT ... 165, 224, 240, 259, 472, 529, 698, 713, 778-779, 781, 784
- SIG_BLOCK ... 781
- SIG_DFL ... 779
- SIG_IGN ... 781
- SIG ... 258-259, 471-473, 475, 672
- SIG_SETMASK ... 781
- SIGSTOP ... 258-259
- SIGTERM ... 165, 259, 471-472, 529, 698, 713
- SIGTTOU ... 617
- SILENT ... 696, 699-700, 702, 714
- Simple Commands ... 216
- single-quote
 - definition of ... 43
- Single-Quotes ... 186
- SINGLE ... 707
- S_IRGRP ... 140, 333, 518, 640
- S_IROTH ... 140, 333, 518, 640
- S_IRUSR ... 140, 333, 518, 526, 640
- S_IRWXG ... 514, 549, 726, 817
- S_IRWXO ... 514, 549, 726, 817
- S_IRWXU ... 360, 514, 549, 557, 726, 817
- S_ISGID ... 34, 333, 336, 362, 436, 522, 551, 661
- S_ISUID ... 34, 333, 336, 362, 436, 522, 551, 661
- S_IWGRP ... 140, 333, 518, 640, 660
- S_IWOTH ... 140, 333, 436, 518, 640, 660
- S_IWUSR ... 140, 333, 518, 526, 640
- S_IXGRP ... 331, 333
- S_IXOTH ... 331, 333
- S_IXUSR ... 331, 333
- slash
 - definition of ... 44
- sleep
 - Suspend execution for an interval
 - ... 603, 842
 - ... 603-605
 - definition of ... 603
- SLR ... 765-766
- Software Development Utilities Option
 - ... 687, 843
- SOH ... 68, 74
- SOH ... 56, 618
- sort ... 5, 29, 179, 467-468, 605-606, 608, 610, 612-613, 668
 - definition of ... 605
- {SORT_LINE_MAX} ... 179
- source code
 - definition of ... 44
- <space>
 - definition of ... 44
- SPEC_CHAR ... 122, 124-125
- Special Built-in Utilities ... 246, 832
- special built-in ... 28, 30, 36, 45, 51, 181-182, 192, 214, 217, 219-220, 230-231, 241, 246, 248, 252-253, 255, 259, 261, 354, 358, 419, 527, 529, 576, 601-602, 653, 679, 832
- Special Control Character Assignments
 - ... 617
- special parameter
 - definition of ... 184
- Special Parameters ... 192
- Special Patterns ... 274
- SPEC_CHAR ... 121
- split ... 417, 837
- SQL ... 8
- ssize_t ... 786
- standard error
 - definition of ... 44
- standard input
 - definition of ... 44
- Standard Libraries ... 730, 820
- standard output
 - definition of ... 44
- standard utilities
 - definition of ... 44
- START ... 616-617, 759
- START/STOP ... 616
- stat ... 640
- stat() ... 32, 35, 138, 155, 553, 801
- st_atime ... 35
- st_ctime ... 35
- STDIN_FILENO ... 783
- <stdio.h> ... 730, 774
- <stdlib.h> ... 730
- STDOUT_FILENO ... 782
- st_gid ... 693
- sticky bit ... 336
- st_mode ... 693
- st_mtime ... 35, 693
- STOP ... 616-617
- strcoll() ... 81
- stream
 - definition of ... 44

- STREAMS ... 49
- strerror()* ... 793
- Strictly Conforming POSIX.2 Application
 - ... 4, 6, 13, 15, 23-24, 104, 109, 113, 115, 128, 135, 166, 175
- String Functions ... 278
- String Operand ... 426
- <string.h> ... 730
- STRING ... 282, 286-287, 303-304
- STRING ... 265, 273, 288, 305
- strip
 - Remove unnecessary information from executable files ... 716, 844
 - ... 716-718
 - definition of ... 716
- strtod()* ... 572-573
- strtol()* ... 572-573
- strtoul()* ... 572-573
- Structure Type *glob_t* ... 800
- Structure Type *regex_t* ... 787
- Structure Type *regmatch_t* ... 787
- Structure Type *wordexp_t* ... 804
- strxfrm()* ... 81
- st_size* ... 693
- stty
 - Set the options for a terminal ... 613, 842
 - ... 8, 47, 613-615, 619-622
 - Circumflex Control Characters ... 618
 - Control Character Names ... 617
 - definition of ... 613
- st_uid* ... 693
- STX ... 68, 74
- STX ... 56, 618
- SUB_ASSIGN ... 282, 286-287
- SUB ... 68, 74
- SUB_ASSIGN ... 290
- SUB ... 56, 618
- SUBSCRIPT-LOWER ... 100
- SUBSEP
 - awk variable ... 273
- SUBSEP ... 270, 293
- subshell
 - definition of ... 184
- Substitute Command ... 413
- SUFFIXES ... 702-704, 707
- SUFFIX ... 707
- sum ... 5, 343, 345
- SUPERScript-LOWER ... 100
- super-user ... 366, 508, 559, 833
- supplementary group ID
 - definition of ... 44
- supplementary group IDs ... 138
- supplementary groups ... 33, 36, 44, 138-139, 462, 464-466
- SUSP ... 616-617
- SVID ... 826
- switch ... 228
- SW_DEV ... 179, 777, 811
- Symbolic Constants for Portability
 - Specifications ... 179
- Symbolic Limits ... 173
- Symbolic Utility Limits ... 174
- SYN ... 68, 74
- SYN ... 56, 618
- sysconf()* ... 174-176, 179-180, 443, 446, 722-723, 775-777, 809-811
 - definition of ... 811
- <sys/stat.h> ... 20
- system documentation
 - definition of ... 24
- System III ... 397, 638
- System V ... 4-5, 9, 49, 181, 187, 208, 213, 232, 245, 254, 256, 261, 315, 320-321, 325, 328, 335-336, 339, 345, 349, 353, 371, 378, 387, 393, 397, 401-402, 418, 437-438, 443, 450, 458, 501, 509-510, 513, 517, 536-537, 557-559, 567, 573, 587, 595-596, 611-612, 621, 626-627, 635-639, 644, 650-651, 677, 681, 692-693, 708-710, 712, 714-715, 734, 747-749, 768, 826, 832, 836-837
- system
 - definition of ... 45
- system()* ... 2, 8, 10, 51, 53, 167, 181, 188, 280, 446, 598, 700, 714, 720, 778-782, 811
 - definition of ... 778
- <sys/types.h> ... 730

T

- <tab>
 - definition of ... 45
- tail
 - Copy the last part of a file ... 623, 842
 - ... 461-462, 623, 625-627
 - definition of ... 623

- tar ... 66, 552, 557-561, 692
- Target Rules ... 701
- TCOS ... 8
- tee
 - Duplicate standard input ... 628, 842
 - ... 628, 630
 - definition of ... 628
- terminal device (see terminal)
- terminal
 - definition of ... 45
- terminal ... 7-8, 22, 26-27, 29, 33-36, 43, 45, 48, 60, 105-106, 109, 138-139, 143, 159, 163, 169, 177, 179, 188, 213, 276, 282, 301, 303, 321-322, 331, 333, 336, 365, 416-418, 441, 475, 500, 503-504, 508-509, 521, 525-526, 528, 561-562, 565, 579-580, 583, 597, 599, 602, 613-614, 618-622, 632, 654-656, 685, 714, 755-756, 758, 764-765, 769, 779, 785, 816, 834-838, 842
- Terminology and General Requirements
 - ... 19, 830
- Terminology ... 23
- TERM**
 - variable ... 105
- TERM ... 258, 260, 476
- test
 - Evaluate expression ... 631, 842
 - ... 142, 182, 206, 436, 508, 547, 631, 633-636, 638-639, 656, 834
 - definition of ... 631
- text column
 - definition of ... 45
- text file ... 31, 38, 45, 47, 108, 136, 160-161, 173-174, 219-221, 265, 296, 301, 351, 369, 371, 389, 403-404, 418, 439, 441-442, 455, 460, 466, 468, 488, 497, 499, 511, 540-542, 554, 564, 577, 587-589, 599, 605, 608, 624, 626, 666, 677, 679, 682, 697, 727-729, 737-738, 751-753, 769, 814, 819, 842
- definition of ... 45
- then ... 229, 247
- Tilde Expansion ... 197
- tilde
 - definition of ... 45
- TILDE ... 100
- time formats ... 88
- time ... 191
- time() ... 640
- <time.h> ... 730
- TIME ... 377, 618
- tm_hour ... 42
- tm_min ... 42
- /tmp ... 109
- TMPDIR**
 - variable ... 105, 109-110, 164, 612, 729, 733, 820
- TMPDIR ... 547
- tmpnam() ... 547
- tm_sec ... 42
- tm_yday ... 42
- tm_year ... 42
- TOC ... 771, 774, 786
- Token Recognition ... 188, 831
- token
 - definition of ... 184
- TOKEN ... 233-235, 759-760
- tolower ... 68, 70, 72, 266, 279, 282, 290, 293, 381, 648-649
- TOSTOP ... 617
- touch
 - Change file access and modification times ... 640, 842
 - ... 142, 155, 508, 640-645, 696
 - definition of ... 640
- toupper ... 68, 70, 72, 99, 266, 279, 282, 290, 293, 381, 648-649
- tr
 - Translate characters ... 645, 842
 - ... 60, 126, 645-647, 650-651
 - definition of ... 645
- trap
 - Trap signals ... 258
 - ... 240, 259
 - definition of ... 258
- Trojan Horse ... 509
- true
 - Return true value ... 652, 842
 - ... 16, 51, 53, 207, 216, 652-653
 - definition of ... 652
- tty
 - Return user's terminal name ... 654, 842
 - ... 654-656
 - definition of ... 654
- ttynam() ... 654-655
- TTY ... 567, 622
- Two-Character Mnemonics ... 885
- TYPE ... 759-760
- typeset ... 231

Typographical Conventions ... 19

TZ

variable ... 105, 373, 376, 378, 389, 504,
565, 641, 643-644

U

UCHAR_MAX ... 537

UCS ... 825

UINT_MAX ... 537

{ULONG_MAX} ... 175

ULONG_MAX ... 537

umask

— Get or set the file mode creation mask
... 657, 842
... 5, 51-52, 240-241, 657-660
definition of ... 657

uname

— Return system name ... 662, 842
... 662-664
definition of ... 662

uname() ... 662, 665

undefined

definition of ... 24

undefined ... 14-15, 24, 42, 48, 57, 61

UNDEFINED ... 96, 101

undefined ... 111-112, 115-116, 118-119, 128,
136-137, 148-151, 160-161, 166, 168, 170,
179, 186, 188, 197, 199, 203, 251, 254, 258,
264, 270, 272, 274-275, 277-279, 281, 288-
289, 291, 296-297, 306, 308, 310, 313-317,
322, 325, 347, 351, 406, 409, 418-419, 431,
438, 445-446, 454, 458, 467, 472, 496, 498,
527, 578, 594, 598, 606, 648-649, 679, 687-
689, 693, 698, 727, 739-740, 742, 744-745,
749, 755, 757, 772, 777, 783-784, 786, 788-
789, 801, 805-806, 884

UNDEFINED ... 78-80

Undo Command ... 414

unfunction ... 261

UNION ... 759

uniq

— Report or filter out repeated lines in a
file ... 665, 842
... 665-669
definition of ... 665

<unistd.h> ... 20, 730, 774, 776-777, 809,
811

UNIX ... 3-10, 48-49, 59, 63, 81, 129, 138,
153-154, 175, 177, 231, 245, 340, 644, 826,
829, 832

unlink() ... 142, 155, 361, 477, 580

unput() ... 746

unset

— Unset values and attributes of variables
and functions ... 260
... 192, 253, 257, 260-261, 357
definition of ... 260

unspecified

definition of ... 24

unspecified ... 14-15, 24, 26-29, 31, 35-38, 44,
47, 55, 62-63, 70, 83-84, 86-88, 91-92, 104-
106, 115, 128, 135, 142, 157, 160-161, 163-
164, 166-167, 176, 178, 181, 183-185, 188,
191, 196-200, 204-205, 209, 211-212, 216-
217, 219, 228, 234, 240, 243, 245, 250, 252,
254, 260, 269, 271-272, 275, 277, 279, 297,
308, 315, 321, 333, 349-350, 360-362, 364-
366, 372, 376-377, 379-380, 382, 386, 391,
398, 407, 411, 413, 416, 418, 425-428, 435,
437, 439, 442, 446, 448-450, 454, 467, 471,
481, 489, 491-492, 496-502, 510-511, 514,
524, 526, 531, 534, 539, 549-550, 552, 555-
556, 563, 570, 582-583, 591-593, 596, 599,
607, 613-615, 618, 620-621, 624, 627, 633,
642, 647-650, 657-658, 660, 666, 671-672,
678-679, 687, 689, 691, 695-698, 701, 703,
711, 713, 716-717, 726-731, 733-737, 742,
745, 747-748, 750, 752-753, 758, 763-764,
773, 778, 782, 789, 795-796, 800, 802, 804,
809, 812, 814, 816-818, 820-823

until ... 227, 229-230, 246, 248, 255

Loop ... 229

UPE ... 52, 108, 143, 162, 182-183, 191, 195,
232, 320, 357, 476, 598, 601-602, 622, 673,
832, 835, 837

UPPER-CASE ... 100-101

UPPER ... 101

UPPER_CASE ... 77

Usage

Examples ... 461, 490, 512, 626

USD ... 101

USENET ... 394

user database

definition of ... 45

user ID

definition of ... 45

effective ... 31

real ... 42

saved set- ... 42

user name

definition of ... 45

User Portability Utilities Option ... 685, 843

user
 definition of ... 48

User-Defined Functions ... 281

user-defined ordering of collating elements
 ... 72

USER
 variable ... 108

USHORT_MAX ... 537

/usr ... 109, 557

/usr/bin ... 109

/usr/lib ... 109

/usr/lib/libc.a ... 734

/usr/lib/libf.a ... 823

/usr/local ... 109

/usr/local/bin ... 155

/usr/man ... 109

/usr/tmp ... 109

ustar ... 558

UTC0 ... 373

UTC ... 378, 641, 644

utilities
 [... 631

 ar ... 687

 asa ... 813

 awk ... 263

 basename ... 297

 bc ... 301

 break ... 246

 c89 ... 726

 case ... 227

 cat ... 318

 cd ... 322

 chgrp ... 326

 chmod ... 329

 chown ... 337

 cksum ... 341

 cmp ... 347

 colon ... 247

 comm ... 350

 command ... 354

 continue ... 248

 cp ... 359

 cut ... 368

 date ... 373

 dd ... 379

 diff ... 388

 dirname ... 395

 dot ... 248

 echo ... 399

 ed ... 402

 env ... 419

 eval ... 249

 exec ... 250

 exit ... 251

 export ... 252

 expr ... 423

 false ... 428

 find ... 430

 fold ... 438

 for ... 226

 fort77 ... 817

 getconf ... 442

 getopts ... 447

 grep ... 452

 head ... 459

 id ... 462

 if ... 228

 join ... 466

 kill ... 471

 lex ... 736

 ln ... 476

 locale ... 480

 localedef ... 486

 logger ... 491

 logname ... 494

 lp ... 496

 ls ... 502

 mailx ... 510

 make ... 695

 mkdir ... 514

 mkfifo ... 518

 mv ... 521

 nohup ... 526

 od ... 530

 paste ... 538

 pathchk ... 543

 pax ... 548

 pr ... 562

 printf ... 568

 pwd ... 574

 read ... 576

 readonly ... 253

 return ... 254

 rm ... 579

 rmdir ... 584

 sed ... 587

 set ... 254

 sh ... 597

 shift ... 258

 sleep ... 603

 sort ... 605

 strip ... 716

 stty ... 613

 tail ... 623

 tee ... 628

 test ... 631

 touch ... 640

 tr ... 645

 trap ... 258

Copyright © 1991 IEEE. All rights reserved.

This is an unapproved IEEE Standards Draft, subject to change.

- true ... 652
- tty ... 654
- umask ... 657
- uname ... 662
- uniq ... 665
- unset ... 260
- wait ... 669
- wc ... 674
- while ... 229
- xargs ... 678
- yacc ... 750
- Utility Argument Syntax ... 147
- Utility Conventions ... 147, 830
- Utility Description Defaults ... 156, 830
- Utility Limit Minimum Values ... 173
- Utility Syntax Guidelines ... 152
- utility
 - definition of ... 45
- utimbuf* ... 640
- utime()* ... 640
- UUCP ... 7, 838

V

- Valid Character Class Combinations ... 71
- validfnam ... 547
- variable assignment [assignment]
 - definition of ... 184
- variable
 - definition of ... 184
- Variable-Length Character Mnemonics ... 886
- Variables and Special Variables ... 271
- VARIABLE ... 261
- Variables ... 194
- VAR ... 427
- VAX-11 ... 827
- VAX ... 9
- VEOF ... 617
- VEOL ... 617
- VERASE ... 617
- Version 7 ... 3, 181, 231, 335, 387, 416, 596, 638
- VERSION ... 173, 811
- <vertical-tab>
 - definition of ... 46
- vi ... 228, 514, 791, 834

- VIII ... 378
- VII ... 378
- VINTR ... 617
- VISUAL**
 - variable ... 108, 261
- VISUAL ... 261
- VKILL ... 617
- VM/CMS ... 3
- VMS ... 3
- VQUIT ... 617
- VSTART ... 617
- VSTOP ... 617
- VSUSP ... 617

W

- wait
 - Await process completion ... 669, 843
 - ... 51-52, 240, 471, 669-673
 - definition of ... 669
- wait()* ... 215, 673, 780, 783, 785
- waitpid()* ... 21, 673, 779-780, 783-785
- wc ... 158, 674-677
 - definition of ... 674
- WEXITSTATUS ... 215
- WG15 ... 49
- while ... 218, 227, 229, 239, 246, 248, 255, 269, 296, 429, 547
 - definition of ... 229
 - Loop ... 229
- white space
 - definition of ... 46
- Word Expansions ... 195, 831
- word
 - definition of ... 184
- wordexp()* ... 722, 803-808
 - definition of ... 804
 - flags* Argument ... 805
 - Return Values ... 806
- <wordexp.h> ... 804-806
- wordexp_t*
 - definition of ... 804
- wordfree()* ... 722, 804-806, 808
- WORD ... 235-238
- WORD ... 233-235, 239
- working directory
 - definition of ... 46

WRDE ... 773
 WRDE_APPEND ... 805
 WRDE_BADCHAR ... 806
 WRDE_BADVAL ... 806
 WRDE_CMDSUB ... 806
 WRDE_DOOFFS ... 805-806
 WRDE_NOCMD ... 805-807
 WRDE_NOSPACE ... 806
 WRDE ... 806
 WRDE_REUSE ... 805, 808
 WRDE_SHOWERR ... 805-808
 WRDE_SYNTAX ... 806
 WRDE_UNDEF ... 805-806
 Write Command ... 414
 write
 definition of ... 46
 write() ... 32, 735, 823

yesexpr ... 26, 92, 102, 363, 434, 485, 523,
 554, 581
 YFLAGS ... 706-707
 y.output ... 750, 767, 769
 y.tab.c ... 750, 767
 y.tab.h ... 713, 750, 767
 YYABORT ... 762
 YYACCEPT ... 762
 YYDEBUG ... 764
 yyerror() ... 750, 753, 763
 YYERROR ... 762
 yylex() ... 739, 741, 744-745, 749-750, 753,
 758, 763-764, 767
 yyparse() ... 750, 752, 763
 YYRECOVERING ... 762
 YYSTYPE ... 752, 756-757, 767
 yywrap() ... 745

X

X/2047 ... 418
 X.400 ... 513
 X.400 ... 513
 xarg ... 683
 xargs
 — Construct argument list(s) and invoke
 utility ... 678, 843
 ... 47, 51, 215, 357, 422, 529, 678-683,
 837
 definition of ... 678
 xd ... 536
 XII ... 378
 X/Open ... 9-10, 82, 131-132, 353, 827, 832
 XPG3 ... 353, 651, 712
 XPG ... 651

Y

yacc
 — Yet another compiler compiler ... 750,
 845
 ... 22, 25, 44, 161, 713, 731, 734, 736,
 750-758, 761, 763-768
 definition of ... 750
 Internal Limits ... 765
 Library ... 763
 YACC ... 706-707

Acknowledgments

We wish to thank the following organizations for donating significant computer, printing, and editing resources to the production of this standard: Amdahl Corporation, AT&T, Concurrent Computer Corporation, the POSIX Software Group, UniSoft Corporation, and the X/Open Group.

This document was also approved by ISO/IEC JTC 1/SC22/WG15 as ISO/IEC 9945-2:199x. The IEEE wishes to thank the advisory groups of the National Bodies participating in WG15 for their contributions: Austria, Belgium, Canada, Denmark, France, Germany, Japan, Netherlands, United Kingdom, USA, and USSR.

The IEEE also wishes to thank the delegates to WG15 for their contributions:

AUSTRIA	Yves Delarue	UK
Gerhard Schmitt	Eric Dumas	Nigel Bevan
Wolfgang Schwabl	Maurice Fathi	Cornelia Boldyreff
	Gerald Krummeck	Dave Cannon
CANADA	Herve Schauer	Don Chacon
Joe Cote	Hubert Zimmerman	Dominic Dunlop
Patrick Dempster		David Flint
George Kriger	GERMANY	Don E. Folland
Bernard Martineau	Ron Elliot	Martin Kirk
Major Douglas J. Moore	Helmut Stiegler	Neil Martin
Arnie Powell	Claus Unger	Brian Meek
Paul Renaud	Rainer Zimmer	Kevin Murphy
Richard Sniderman		Ian Newman
	IRELAND	Philip Rushton
CEC	Hans-Jurgen Kugler	
Phil Bertrand		USA
Manuel Carbajo	JAPAN	Robert Bismuth
Michel Colin	Hiromichi Kogure	Steven L. Carter
	Shigekatsu Nakao	Terence S. Dowling
DENMARK	Yasushi Nakahara	Ron Elliott
Peter E. Cordsen	Nobuo Saito	Dale Harris
Isak Korn		John Hill
Keld Simonsen	NETHERLANDS	James D. Isaak
Claus Tondering	J. Van Katwijk	Hal Jespersen
	Willem Wakker	Roger J. Martin
FINLAND	H.J. Weegenaar	Shane McCarron
Jikka Haikala		Barry Needham
	SWEDEN	Donn S. Terry
FRANCE	Mat Linder	Alan Weaver
Pascal Beyles		
Christophe Binot		USSR
Claude Bourstin		V. Koukhar
Jean-Michel Cornu		Ostapenko Georgy Pavlovich

Also we wish to thank the organizations employing the members of the Working Group and the Balloting Group for both covering the expenses related to attending and participating in meetings, and donating the time required both in and out of meetings for this effort.

3M Company
ACE Associated Computer Experts b.v.
Aeon Technologies, Inc.

Mallinckrodt Institute
Martin Marietta Data Systems
Masscomp

Alis Development	Mercury Computer Systems
Amdahl Corporation*	Microsoft Corporation
Apollo Computer Inc.	Mindcraft, Inc.
Apple Computer Inc.	Mitsubishi Electric Corporation
Ardent Computer	Mortice Kern Systems Inc.
AT&T*	Motorola Inc.
AT&T Bell Laboratories	Myrias Research Corporation
AT&T UNIX Pacific Co., Ltd.	NAPS International
Axon Data Information Systems	NASA-KSC
BBN Communications Corporation	National Institute of Standards and Technology
Bell Communications Research	Naval Postgraduate School
BNR, Inc.	NCR Corporation*
Bolt Beranek & Newman	Northern Telecom, Inc.
BP Research International	Novell, Inc.
British Telecom	Ohio State University
British Telecom Research Labs	Pacific Marine Tech
Charles River Data Systems*	POSIX Software Group
Chemical Abstracts Service	PRC
Chorus Systemes	Prime Computer, Inc.
Commission of the European Communities	R&D Associates
Computer X, Inc.	Rabbit Software Corporation
Concurrent Computer Corporation*	ROLM Mil Spec Computers
Control Data Corporation*	Sandia National Laboratory*
Convergent Technologies	Santa Cruz Operation Inc.
Convex Computer Corporation	Saudia National Labs
Cray Research, Inc.*	Sequent
Cyber-Dyne, Inc.	Shia Systems Inc.
Datapoint Corporation	Simpact Associates, Inc.
Data General Corporation	SoHar
Digital Equipment Corporation*	Sphinx Ltd.
Digital ETU	St. Lawrence College
Douglas Aircraft Company	Stellar Computer Inc.
Electrospace Systems, Inc.	Sun Microsystems, Inc.*
Emerging Technology Group Inc.	Syntactics
Emory University Computing Center	Tandem Computers Inc.
Encore Computer	Technical University of Vienna
ETA Systems, Inc.	Tektronix, Inc.*
Federal Judicial Center	Texas Instruments
Ford Motor Company	The Instruction Set Ltd.
Free Software Foundation	The MITRE Corp
General Electric Corporation	Toshiba Corporation
Georgia Institute of Technology	UFPb-GRC
Gilbert International Inc.	UniForum*
Gould CSD	UniSoft Corporation
Harris Corporation	UniSoft Ltd.
HCR Corporation	Unisys Corporation*
Hewlett-Packard Company*	University Of California, Berkeley
Honeywell Bull, Inc.	University of Hong Kong
HQ USAISC	University of Indonesia
Hughes Aircraft Co.	University of Maryland
IBM Corporation*	University of Texas at Arlington
IBM Japan	University of Utah
IBM Systems Integration Division	University of Victoria
Icon International, Inc.	University of Vienna
Intel	USAF
Interactive Systems Corporation	USAISEC
Ironwood Software	USENIX Association*
KAIST	US Army

Lachman Associates, Inc.
Lawrence Livermore National Lab
Loral Rolm MilSpec Computers
Lotus Development Corporation
Mahavishi International University
Whitesmiths, Ltd.
Woods Hole Oceanographic Inst.

US Army Ballistic Research Lab.
US Army Computer Engineering Center
US West Advanced Technologies
Videoton
Wang Laboratories
X/Open Company Ltd.
XIOS Systems Corporation

In the preceding list, the organizations marked with an asterisk (*) have hosted 1003 Working Group meetings since the group's inception in 1985, providing useful logistical support for the ongoing work of the committees.