

INTERNATIONAL
STANDARD

**ISO/IEC
9945-2**

**ANSI/IEEE
Std 1003.2**

First edition
1993-12-22

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

**Part 2:
Shell and Utilities**

Volume 2

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

Partie 2: Enveloppe et services

Tome 2



Reference number
ISO/IEC 9945-2:1993(E)
ANSI/IEEE
Std 1003.2-1993

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

Copyright © 1993 by the
Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1993
Printed in the United States of America

ISBN 1-55937-255-9

*No part of this publication may be reproduced in any form,
in an electronic retrieval system or otherwise,
without the prior written permission of the publisher.*

December 22, 1993

SH15628

International Standard ISO/IEC 9945-2: 1993
ANSI/IEEE Std 1003.2-1992
(Includes ANSI/IEEE Std 1003.2a-1992)

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

Sponsor

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

Approved September 17, 1992

IEEE Standards Board

Approved April 5, 1993

American National Standards Institute

Approved 1993 by the

**International Organization for Standardization
and by the
International Electrotechnical Commission**

Abstract: This standard 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. When the User Portability Utilities Option is included, the standard also defines a common environment for general-purpose time-sharing users on character-oriented display terminals.

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



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



Published by
The Institute of Electrical and Electronics Engineers, Inc.



Foreword

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

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 9945-2:1993 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, subcommittee 22, *Programming languages, their environments and system software interfaces*.

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

- *Part 1: System application program interface (API) [C language]*
- *Part 2: Shell and Utilities*
- *Part 3: System administration* (under development)

Annexes A, B and C form an integral part of this part of ISO/IEC 9945. Annexes D, E, F, G and H are for information only.



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

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
Foreword	ix
Introduction	x
Organization of This Part of ISO/IEC 9945	xi
Base Documents	xii
Related Standards Activities	xiii
Section 1: General	1
1.1 Scope	1
1.2 Normative References	3
1.3 Conformance	4
1.4 Test Methods	7
Section 2: Terminology and General Requirements	9
2.1 Conventions	9
2.2 Definitions	13
2.3 Built-In Utilities	39
2.4 Character Set	40
2.5 Locale	44
2.6 Environment Variables	73
2.7 Required Files	77
2.8 Regular Expression Notation	77
2.9 Dependencies on Other Standards	92
2.10 Utility Conventions	97
2.11 Utility Description Defaults	101
2.12 File Format Notation	108
2.13 Configuration Values	111
2.14 Terminal Characteristics	114
Section 3: Shell Command Language	117
3.1 Shell Introduction	117
3.2 Quoting	117
3.3 Token Recognition	119
3.4 Reserved Words	121
3.5 Parameters and Variables	122
3.6 Word Expansions	125
3.7 Redirection	130
3.8 Exit Status and Errors	133
3.9 Shell Commands	135
3.10 Shell Grammar	143
3.11 Signals and Error Handling	149
3.12 Shell Execution Environment	149
3.13 Pattern Matching Notation	150

3.14	Special Built-In Utilities	152
3.14.1	break – Exit from for, while, or until loop	153
3.14.2	colon – Null utility	153
3.14.3	continue – Continue for, while, or until loop	153
3.14.4	dot – Execute commands in current environment	154
3.14.5	eval – Construct command by concatenating arguments	154
3.14.6	exec – Execute commands and open, close, and/or copy file descriptors	154
3.14.7	exit – Cause the shell to exit	155
3.14.8	export – Set export attribute for variables	155
3.14.9	readonly – Set read-only attribute for variables	155
3.14.10	return – Return from a function	156
3.14.11	set – Set/unset options and positional parameters	156
3.14.12	shift – Shift positional parameters	159
3.14.13	trap – Trap signals	160
3.14.14	unset – Unset values and attributes of variables and functions	161
	Section 4: Execution Environment Utilities	163
4.1	awk – Pattern scanning and processing language	163
4.2	basename – Return nondirectory portion of pathname	190
4.3	bc – Arbitrary-precision arithmetic language	192
4.4	cat – Concatenate and print files	205
4.5	cd – Change working directory	207
4.6	chgrp – Change file group ownership	210
4.7	chmod – Change file modes	212
4.8	chown – Change file ownership	217
4.9	cksum – Write file checksums and sizes	220
4.10	cmp – Compare two files	222
4.11	comm – Select or reject lines common to two files	225
4.12	command – Execute a simple command	228
4.13	cp – Copy files	231
4.14	cut – Cut out selected fields of each line of a file	237
4.15	date – Write the date and time	240
4.16	dd – Convert and copy a file	244
4.17	diff – Compare two files	249
4.18	dirname – Return directory portion of pathname	254
4.19	echo – Write arguments to standard output	257
4.20	ed – Edit text	259
4.21	env – Set environment for command invocation	272
4.22	expr – Evaluate arguments as an expression	274
4.23	false – Return false value	278
4.24	find – Find files	279
4.25	fold – Fold lines	285
4.26	getconf – Get configuration values	287
4.27	getopts – Parse utility options	291
4.28	grep – File pattern searcher	295
4.29	head – Copy the first part of files	299
4.30	id – Return user identity	302

4.31	join – Relational database operator	305
4.32	kill – Terminate or signal processes	309
4.33	ln – Link files	313
4.34	locale – Get locale-specific information	315
4.35	localedef – Define locale environment	320
4.36	logger – Log messages	323
4.37	logname – Return the login name of the user	325
4.38	lp – Send files to a printer	327
4.39	ls – List directory contents	330
4.40	mailx – Process messages	336
4.41	mkdir – Make directories	358
4.42	mkfifo – Make FIFO special files	360
4.43	mv – Move files	363
4.44	nohup – Invoke a utility immune to hangups	367
4.45	od – Dump files in various formats	369
4.46	paste – Merge corresponding or subsequent lines of files	374
4.47	pathchk – Check pathnames	377
4.48	pax – Portable archive interchange	380
4.49	pr – Print files	388
4.50	printf – Write formatted output	393
4.51	pwd – Return working directory name	396
4.52	read – Read a line from standard input	398
4.53	rm – Remove directory entries	401
4.54	rmdir – Remove directories	404
4.55	sed – Stream editor	406
4.56	sh – Shell, the standard command language interpreter	414
4.57	sleep – Suspend execution for an interval	429
4.58	sort – Sort, merge, or sequence check text files	431
4.59	stty – Set the options for a terminal	437
4.60	tail – Copy the last part of a file	444
4.61	tee – Duplicate standard input	447
4.62	test – Evaluate expression	449
4.63	touch – Change file access and modification times	453
4.64	tr – Translate characters	457
4.65	true – Return true value	462
4.66	tty – Return the terminal name of the user	464
4.67	umask – Get or set the file mode creation mask	466
4.68	uname – Return system name	469
4.69	uniq – Report or filter out repeated lines in a file	472
4.70	wait – Await process completion	474
4.71	wc – Word, line, and byte/character count	477
4.72	xargs – Construct argument list(s) and invoke utility	480
Section 5: User Portability Utilities Option		485
5.1	alias – Define or display aliases	485
5.2	at – Execute commands at a later time	488
5.3	batch – Execute commands when the system load permits	495
5.4	bg – Run jobs in the background	496
5.5	crontab – Schedule periodic background work	499
5.6	csplit – Split files based on context	502

5.7	ctags – Create a tags file	505
5.8	df – Report free disk space	509
5.9	du – Estimate file space usage	512
5.10	ex – Text editor	515
5.11	expand – Convert tabs to spaces	542
5.12	fc – Process command history list	545
5.13	fg – Run jobs in the foreground	550
5.14	file – Determine file type	552
5.15	jobs – Display status of jobs in the current session	555
5.16	man – Display system documentation	558
5.17	mesg – Permit or deny messages	561
5.18	more – Display files on a page-by-page basis	563
5.19	newgrp – Change to a new group	574
5.20	nice – Invoke a utility with an altered system scheduling priority	578
5.21	nm – Write the name list of an object file	580
5.22	patch – Apply changes to files	584
5.23	ps – Report process status	589
5.24	renice – Set system scheduling priorities of running processes	594
5.25	split – Split files into pieces	597
5.26	strings – Find printable strings in files	600
5.27	tabs – Set terminal tabs	603
5.28	talk – Talk to another user	606
5.29	time – Time a simple command	609
5.30	tput – Change terminal characteristics	612
5.31	unalias – Remove alias definitions	614
5.32	unexpand – Convert spaces to tabs	616
5.33	uudecode – Decode a binary file	619
5.34	uuencode – Encode a binary file	621
5.35	vi – Screen-oriented (visual) display editor	624
5.36	who – Display who is on the system	655
5.37	write – Write to another user	657
Section 6: Software Development Utilities Option		661
6.1	ar – Create and maintain library archives	661
6.2	make – Maintain, update, and regenerate groups of programs	666
6.3	strip – Remove unnecessary information from executable files	679
Section 7: Language-Independent System Services		683
7.1	Shell Command Interface	683
7.2	Access Environment Variables	684
7.3	RE Matching	684
7.4	Pattern Matching	684
7.5	Command Option Parsing	684
7.6	Generate Pathnames Matching a Pattern	685
7.7	Perform Word Expansions	685
7.8	Get POSIX Configurable Variables	685
7.9	Locale Control	686
Annex A (normative) C-Language Development Utilities Option		687

A.1	c89 – Compile Standard C programs	687
A.2	lex – Generate programs for lexical tasks	693
A.3	yacc – Yet another compiler compiler	703
Annex B (normative) C-Language Bindings Option		719
B.1	C-Language Definitions	719
B.2	C Numeric Limits	721
B.3	C Binding for Shell Command Interface	723
B.4	C Binding for Access Environment Variables	726
B.5	C Binding for RE Matching	726
B.6	C Binding for Match Filename or Pathname	731
B.7	C Binding for Command Option Parsing	732
B.8	C Binding for Generate Pathnames Matching a Pattern	733
B.9	C Binding for Perform Word Expansions	736
B.10	C Binding for Get POSIX Configurable Variables	739
B.11	C Binding for Locale Control	741
Annex C (normative) FORTRAN Development and Runtime Utilities		
	Options	743
C.1	asa – Interpret carriage-control characters	743
C.2	fort77 – FORTRAN compiler	746
Annex D (informative) Bibliography		751
Annex E (informative) Rationale and Notes		755
E.1	General	755
E.2	Terminology and General Requirements	769
E.3	Shell Command Language	822
E.4	Execution Environment Utilities	858
E.5	User Portability Utilities Option	971
E.6	Software Development Utilities Option	1011
E.7	Language-Independent System Services	1021
E.8	C-Language Development Utilities Option	1022
E.9	C-Language Bindings Option	1033
E.10	FORTRAN Development and Runtime Utilities Options	1049
Annex F (informative) Portability Considerations		1053
F.1	User Requirements	1053
F.2	Portability Capabilities	1056
F.3	Profiling Considerations	1059
Annex G (informative) Sample National Profile		1063
G.1	Danish National Profile Example	1063
G.2	Additional Danish Utility Requirements	1064
G.3	Locale String Definition Guideline	1068
G.4	Scope of Danish National Locale	1068
G.5	Character Mnemonics Guidelines	1149
G.6	Danish Charmap File Examples	1152
Annex H (informative) Proposals for Future Revisions		1193

H.1	Resolve International Comments to DIS 9945-2	1193
H.2	Resolve Interpretation Requests and Correct Technical Defects	1196
H.3	Revise Granularity of Options	1197
H.4	Support Features of 9945-1 Revision	1197
H.5	Reorganize 9945-1 and 9945-2 Material	1198
Identifier Index		1199
Alphabetic Topical Index		1203

FIGURES

Figure E-1	– bc Exponential Example	874
Figure E-2	– Shell Script Simulating cat s	875
Figure E-3	– ASCII to EBCDIC Conversion	895
Figure E-4	– ASCII to IBM EBCDIC Conversion	896
Figure E-5	– getopt Example	911
Figure E-6	– nice Implementation Example	995
Figure E-7	– lex Example Program	1029
Figure E-8	– Sample <i>system()</i> Implementation	1038
Figure E-9	– Sample <i>pclose()</i> Implementation	1040
Figure E-10	– Example RE Matching	1041
Figure E-11	– Argument Processing With <i>getopt()</i>	1044

TABLES

Table 2-1	– Typographical Conventions	9
Table 2-2	– Job Control Job ID Formats	27
Table 2-3	– Regular Built-in Utilities	39
Table 2-4	– Character Set and Symbolic Names	40
Table 2-5	– Control Character Set	42
Table 2-6	– LC_CTYPE Category in the POSIX Locale	49
Table 2-7	– Valid Character Class Combinations	52
Table 2-8	– LC_COLLATE Category in the POSIX Locale	54
Table 2-9	– LC_MONETARY Category in the POSIX Locale	60
Table 2-10	– LC_NUMERIC Category in the POSIX Locale	63
Table 2-11	– LC_TIME Category in the POSIX Locale	65
Table 2-12	– LC_MESSAGES Category in the POSIX Locale	67
Table 2-13	– BRE Precedence	83
Table 2-14	– ERE Precedence	87
Table 2-15	– C Standard Operators and Functions	96
Table 2-16	– Escape Sequences	109
Table 2-17	– Utility Limit Minimum Values	112
Table 2-18	– Symbolic Utility Limits	113
Table 2-19	– Optional Facility Configuration Values	114
Table 2-20	– Control Character Names	115

Table 3-1	– Shell Error Summary	134
Table 4-1	– awk Expressions in Decreasing Precedence	168
Table 4-2	– awk Escape Sequences	188
Table 4-3	– bc Operators	195
Table 4-4	– chmod Octal Mode Values	216
Table 4-5	– expr Expressions	277
Table 4-6	– od Named Characters	373
Table 4-7	– stty Control Character Names	441
Table 4-8	– stty Circumflex Control Characters	442
Table 5-1	– file Output Strings	554
Table 5-2	– ps Variable Names and Default Headers	593
Table 7-1	– POSIX.1 Numeric-Valued Configurable Variables	686
Table A-1	– lex Table Size Declarations	697
Table A-2	– lex Escape Sequences	699
Table A-3	– lex ERE Precedence	700
Table A-4	– yacc Internal Limits	717
Table B-1	– POSIX.2 Reserved Header Symbols	720
Table B-2	– _POSIX_C_SOURCE	720
Table B-3	– C Macros for Symbolic Limits	722
Table B-4	– C Compile-Time Symbolic Constants	722
Table B-5	– C Execution-Time Symbolic Constants	723
Table B-6	– Structure Type <i>regex_t</i>	727
Table B-7	– Structure Type <i>regmatch_t</i>	727
Table B-8	– <i>regcomp()</i> <i>cflags</i> Argument	727
Table B-9	– <i>regexexec()</i> <i>eflags</i> Argument	727
Table B-10	– <i>regcomp()</i> , <i>regexexec()</i> Return Values	730
Table B-11	– <i>fnmatch()</i> <i>flags</i> Argument	731
Table B-12	– Structure Type <i>glob_t</i>	734
Table B-13	– <i>glob()</i> <i>flags</i> Argument	735
Table B-14	– <i>glob()</i> Error Return Values	736
Table B-15	– Structure Type <i>wordexp_t</i>	737
Table B-16	– <i>wordexp()</i> <i>flags</i> Argument	737
Table B-17	– <i>wordexp()</i> Return Values	739
Table B-18	– <i>confstr()</i> <i>name</i> Values	739
Table B-19	– C Bindings for Numeric-Valued Configurable Variables	740
Table E-1	– Parameter Expansion Summary	831
Table E-2	– <i>dirname</i> Examples	899
Table E-3	– Optional Utility Combinations	971
Table F-1	– POSIX.2 Portability Capability Summary	1056
Table G-1	– Danish Locale Options	1064

Annex C (normative)

FORTRAN Development and Runtime Utilities Options

1 This annex describes utilities used for the development of FORTRAN language
2 applications, including compilation or translation of FORTRAN source code, and
3 the execution of certain FORTRAN applications at runtime.

4 The utilities described in this annex may be provided by the conforming system;
5 however, any system claiming conformance to the FORTRAN Development Utili-
6 ties Option shall provide the `fort77` utility and any system claiming confor-
7 mance to the FORTRAN Runtime Utilities Option shall provide the `asa` utility.

8 C.1 `asa` – Interpret carriage-control characters

9 This utility is optional. It shall be provided on systems that support the FOR-
10 TRAN Runtime Utilities Option.

11 C.1.1 Synopsis

12 `asa [file ...]`

13 C.1.2 Description

14 The `asa` utility shall write its input files to standard output, mapping carriage-
15 control characters from the text files to line-printer control sequences in an
16 implementation-defined manner.

17 The first character of every line shall be removed from the input, and the follow-
18 ing actions shall be performed:

19 If the character removed is

20	<code><space></code>	The rest of the line shall be output without change.
21	<code>0</code>	A <code><newline></code> shall be output, then the rest of the input line.
22	<code>1</code>	One or more implementation-defined characters that cause an
23		advance to the next page shall be output, followed by the rest of
24		the input line.

25 + The <newline> of the previous line shall be replaced with one or
26 more implementation-defined characters that causes printing to
27 return to column position 1, followed by the rest of the input line.
28 If the + is the first character in the input, it shall have the same
29 effect as <space>.

30 The action of the `asa` utility is unspecified upon encountering any character other
31 than those listed above as the first character in a line.

32 **C.1.3 Options**

33 None.

34 **C.1.4 Operands**

35 *file* A pathname of a text file used for input. If no *file* operands are
36 specified, the standard input shall be used.

37 **C.1.5 External Influences**

38 **C.1.5.1 Standard Input**

39 The standard input shall be used only if no *file* operands are specified. See
40 C.1.5.2.

41 **C.1.5.2 Input Files**

42 The input files shall be text files.

43 **C.1.5.3 Environment Variables**

44 The following environment variables shall affect the execution of `asa`:

45 **LANG** This variable shall determine the locale to use for the
46 locale categories when both **LC_ALL** and the correspond-
47 ing environment variable (beginning with **LC_**) do not
48 specify a locale. See 2.6.

49 **LC_ALL** This variable shall determine the locale to be used to over-
50 ride any values for locale categories specified by the set-
51 tings of **LANG** or any environment variables beginning
52 with **LC_**.

53 **LC_CTYPE** This variable shall determine the locale for the interpreta-
54 tion of sequences of bytes of text data as characters (e.g.,
55 single- versus multibyte characters in arguments and
56 input files).

57 **LC_MESSAGES** This variable shall determine the language in which mes-
58 sages should be written.

PART 2: SHELL AND UTILITIES

59 **C.1.5.4 Asynchronous Events**

60 Default.

61 **C.1.6 External Effects**

62 **C.1.6.1 Standard Output**

63 The standard output shall be the text from the input file modified as described in
64 C.1.2.

65 **C.1.6.2 Standard Error**

66 None.

67 **C.1.6.3 Output Files**

68 None.

69 **C.1.7 Extended Description**

70 None.

71 **C.1.8 Exit Status**

72 The `asa` utility shall exit with one of the following values:

73 0 All input files were output successfully.

74 >0 An error occurred.

75 **C.1.9 Consequences of Errors**

76 Default.

77 C.2 `fort77` – FORTRAN compiler

78 This utility is optional. It shall be provided on systems that support the FOR-
79 TRAN Development Utilities Option.

80 C.2.1 Synopsis

81 `fort77 [-c][-g][-L directory]... [-O optlevel][-o outfile][-s][-w]`
82 `operand ...`

83 C.2.2 Description

84 The `fort77` utility is the interface to the FORTRAN compilation system; it shall
85 accept the full FORTRAN language defined by ANSI X3.9-1978 [9]. The system
86 conceptually consists of a compiler and link editor. The files referenced by
87 *operands* are compiled and linked to produce an executable file. (It is unspecified
88 whether the linking occurs entirely within the operation of `fort77`; some systems
89 may produce objects that are not fully resolved until the file is executed.)

90 If the `-c` option is present, for all pathname operands of the form *file.f*, the files

91 `$(basename pathname .f).o`

92 shall be created or overwritten as the result of successful compilation. If the `-c`
93 option is not specified, it is unspecified whether such `.o` files are created or
94 deleted for the *file.f* operands.

95 If there are no options that prevent link editing (such as `-c`) and all operands
96 compile and link without error, the resulting executable file shall be written into
97 the file named by the `-o` option (if present) or to the file `a.out`. The executable
98 file shall be created as specified in 2.9.1.4, except that the file permissions shall be
99 set to

100 `S_IRWXO | S_IRWXG | S_IRWXU`

101 (see Section 5.6.1.2 of POSIX.1 [8]) and that the bits specified by the *umask* of the
102 process shall be cleared.

103 C.2.3 Options

104 The `fort77` utility shall conform to the utility argument syntax guidelines
105 described in 2.10.2, except that:

- 106 — The `-l library` operands have the format of options, but their position
107 within a list of operands affects the order in which libraries are searched.
- 108 — The order of specifying the multiple `-L` options is significant.
- 109 — Conforming applications shall specify each option separately; that is,
110 grouping option letters (e.g., `-cg`) need not be recognized by all
111 implementations.

112 The following options shall be supported by the implementation:

- 113 -c Suppress the link-edit phase of the compilation, and do not
114 remove any object files that are produced.
- 115 -g Produce symbolic information in the object or executable files; the
116 nature of this information is unspecified and may be modified by
117 implementation-defined interactions with other options.
- 118 -s Produce object and/or executable files from which symbolic and
119 other information not required for proper execution using the
120 POSIX.1 {8} *exec* family has been removed (stripped). If both -g
121 and -s options are present, the action taken is unspecified.
- 122 -o *outfile* Use the pathname *outfile*, instead of the default *a.out*, for the
123 executable file produced. If the -o option is present with -c, the
124 result is unspecified.
- 125 -L *directory* Change the algorithm of searching for the libraries named in -l
126 operands to look in the directory named by the *directory* path-
127 name before looking in the default locations. Directories named
128 in -L options shall be searched in the specified order. Implemen-
129 tations shall support at least ten instances of this option in a sin-
130 gle *fort77* command invocation. If a directory specified by a -L
131 option contains a file named *libf.a*, the results are unspecified.
- 133 -O *optlevel* Specify the level of code optimization. If the *optlevel* option-
134 argument is the digit 0, all special code optimizations shall be
135 disabled. If it is the digit 1, the nature of the optimization is
136 unspecified. If the -O option is omitted, the nature of the default
137 optimization provided by the system is unspecified. It is
138 unspecified whether code generated in the presence of the -O 0
139 option is the same as that generated when -O is omitted. Other
140 *optlevel* values may be supported.
- 141 -w Suppress warnings.

142 Multiple instances of -L options can be specified.

143 C.2.4 Operands

144 An *operand* is either in the form of a pathname or the form -l *library*. At least
145 one operand of the pathname form shall be specified. The following operands
146 shall be supported by the implementation:

- 147 *file.f* The pathname of a FORTRAN source file to be compiled and
148 optionally passed to the link editor. The file name operand shall
149 be of this form if the -c option is used.
- 150 *file.a* A library of object files typically produced by *ar* (see 6.1) and
151 passed directly to the link editor. Implementations may recog-
152 nize implementation-defined suffixes other than *.a* as denoting
153 object file libraries.

154 *file.o* An object file produced by `fort77 -c` and passed directly to the
155 link editor. Implementations may recognize implementation-
156 defined suffixes other than `.o` as denoting object files.

157 `-l library` (The letter ell.) Search the library named

158 `liblibrary.a`

159 A library is searched when its name is encountered, so the place-
160 ment of a `-l` operand is significant. Several standard libraries
161 can be specified in this manner, as described in C.2.7. Implemen-
162 tations may recognize implementation-defined suffixes other than
163 `.a` as denoting libraries.

164 The processing of other operands is implementation defined.

165 **C.2.5 External Influences**

166 **C.2.5.1 Standard Input**

167 None.

168 **C.2.5.2 Input Files**

169 The input file shall be one of the following: a text file containing FORTRAN source
170 code; an object file in the format produced by `fort77 -c`; or a library of object
171 files, in the format produced by archiving zero or more object files, using `ar`.
172 Implementations may supply additional utilities that produce files in these for-
173 mats. Additional input files are implementation defined.

174 A `<tab>` character encountered within the first six characters on a line of source
175 code shall cause the compiler to interpret the following character as if it were the
176 seventh character on the line (i.e., in column 7).

177 **C.2.5.3 Environment Variables**

178 The following environment variables shall affect the execution of `fort77`:

179 **LANG** This variable shall determine the locale to use for the
180 locale categories when both **LC_ALL** and the correspond-
181 ing environment variable (beginning with **LC_**) do not
182 specify a locale. See 2.6.

183 **LC_ALL** This variable shall determine the locale to be used to over-
184 ride any values for locale categories specified by the set-
185 tings of **LANG** or any environment variables beginning
186 with **LC_**.

187 **LC_CTYPE** This variable shall determine the locale for the interpreta-
188 tion of sequences of bytes of text data as characters (e.g.,
189 single- versus multibyte characters in arguments and
190 input files).

191 **LC_MESSAGES** This variable shall determine the language in which mes-
192 sages should be written.

193 **TMPDIR** This variable shall be interpreted as a pathname that
194 should override the default directory for temporary files, if
195 any.

196 **C.2.5.4 Asynchronous Events**

197 Default.

198 **C.2.6 External Effects**

199 **C.2.6.1 Standard Output**

200 None.

201 **C.2.6.2 Standard Error**

202 Used only for diagnostic messages. If more than one file operand ending in `.f` (or
203 possibly other unspecified suffixes) is given, for each such file

204 "`%s: \n`", *<file>*

205 may be written to allow identification of the diagnostic message with the
206 appropriate input file.

207 This utility may produce warning messages about certain conditions that do not
208 warrant returning an error (nonzero) exit value.

209 **C.2.6.3 Output Files**

210 Object files, listing files, and/or executable files shall be produced in unspecified
211 formats.

212 **C.2.7 Extended Description**

213 **C.2.7.1 Standard Libraries**

214 The `fort77` utility shall recognize the following `-l` operand for the standard
215 library:

216 `-l f` This library contains all library functions referenced in
217 ANSI X3.9-1978 [9]. An implementation shall not require this
218 operand to be present to cause a search of this library.

219 In the absence of options that inhibit invocation of the link editor, such as `-c`, the
220 `fort77` utility shall cause the equivalent of a `-l f` operand to be passed to the
221 link editor as the last `-l` operand, causing it to be searched after all other object
222 files and libraries are loaded.

223 It is unspecified whether the library `libf.a` exists as a regular file. The imple-
224 mentation may accept as `-l` operands names of objects that do not exist as regu-
225 lar files.

226 C.2.7.2 External Symbols

227 The FORTRAN compiler and link editor shall support the significance of external
228 symbols up to a length of at least 31 B. The compiler may fold case (i.e., may
229 ignore uppercase/lowercase distinctions between identifiers). The action taken
230 upon encountering symbols exceeding the implementation-defined maximum sym-
231 bol length is unspecified.

232 The compiler and link editor shall support a minimum of 511 external symbols
233 per source or object file, and a minimum of 4095 external symbols total. A diag-
234 nostic message is written to standard output if the implementation-defined limit
235 is exceeded; other actions are unspecified.

236 C.2.8 Exit Status

237 The `fort77` utility shall exit with one of the following values:

238	0	Successful compilation or link edit.
239	>0	An error occurred.

240 C.2.9 Consequences of Errors

241 When `fort77` encounters a compilation error, it shall write a diagnostic to stan-
242 dard error and continue to compile other source code operands. It shall return a
243 nonzero exit status, but it is implementation defined whether an object module is
244 created. If the link edit is unsuccessful, a diagnostic message shall be written to
245 standard error, and `fort77` shall exit with a nonzero status.

Annex D (informative) Bibliography

- 1 {B1} ISO 639: 1988, *Code for the representation of names of languages*.¹⁾
2 {B2} ISO 1539: 1991, *Information technology—Programming languages—*
3 *Fortran*.
4 {B3} ISO 2022: 1986, *Information processing—ISO 7-bit and 8-bit coded charac-*
5 *ter sets—Code extension techniques*.
6 {B4} ISO 2047: 1975, *Information processing—Graphical representations for the*
7 *control characters of the 7-bit coded character set*.
8 {B5} ISO 2375: 1985, *Data processing—Procedure for registration of escape*
9 *sequences*.
10 {B6} ISO 3166: 1988, *Codes for the representation of names of countries*.
11 {B7} ISO 6429: 1988, *Information processing—Control functions for 7-bit and*
12 *8-bit coded character sets*.
13 {B8} ISO 6937-2: 1983, *Information processing—Coded character sets for text*
14 *communication—Part 2: Latin alphabetic and non-alphabetic graphic*
15 *characters*.
16 {B9} ISO 8801: 1988, *Data elements and interchange formats—Information*
17 *interchange—Representation of dates and times*.
18 {B10} ISO/IEC 8802-3: 1992, *Information technology—Local and metropolitan area*
19 *networks—Part 3: Carrier sense multiple access with collision detection*
20 *(CSMA/CD) access method and physical layer specifications*.²⁾
21 {B11} ISO 8859, *Information processing—8-bit single-byte coded graphic character*
22 *sets*. (Parts 1 to 9 published.)
23 {B12} ISO/IEC 10367: 1991, *Information processing—Standardized coded graphic*
24 *character sets for use in 8-bit codes*.

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

27 2) IEC documents can be obtained from the IEC office, 3, rue de Varembé, Case Postale 131,
28 CH-1211, Genève 20, Switzerland/Suisse.

- 29 {B13} ISO/IEC 10646-1: 1993, *Information technology—Universal Multiple-Octet*
30 *Coded Character Set (UCS)—Part 1: Architecture and Basic Multilingual*
31 *Plane*.
- 32 {B14} ISO/IEC TR 10000-1: 1992, *Information technology—Framework and taxon-*
33 *omy of International Standardized Profiles—Part 1: Framework*.
- 34 {B15} ISO/IEC JTC 1 N1335, *Final Report of ISO/IEC JTC 1 TSG-1 on Standards*
35 *necessary to define Interfaces for Application Portability (IAP)*, April 1991.
- 36 {B16} International Organization for Standardization/Association Française de
37 Normalisation. *Dictionary of Computer Science/Dictionnaire de*
38 *L'Informatique*. Geneva/Paris: ISO/AFNOR, 1989.
- 39 {B17} ANSI X3.43-1986, *American National Standard Representations of Local*
40 *Time of the Day for Information Interchange*.³⁾
- 41 {B18} CCITT Recommendation X.400, *Message handling system and service*
42 *overview*.⁴⁾
- 43 {B19} GB 2312-1980, China State Bureau of Standards. *Coded Chinese Graphic*
44 *Character Set for Information Interchange*.⁵⁾
- 45 {B20} IEEE Std 100-1992, *The New IEEE Standard Dictionary of Electrical and*
46 *Electronics Terms (ANSI)*.
- 47 {B21} IEEE Std 754-1985 (Reaff. 1991), *IEEE Standard for Binary Floating Point*
48 *Arithmetic*.
- 49 {B22} P1003.0/D15,⁶⁾ *Draft Guide to the POSIX Open Systems Environment*.
- 50 {B23} IEEE Std 1003.3-1991, *IEEE Standard for Information Technology—Test*
51 *Methods for Measuring Conformance to POSIX (ANSI)*.
- 52 {B24} IEEE Std 1003.9-1992, *IEEE Standard for Information Technology—POSIX*
53 *FORTTRAN 77 Language Interfaces—Part 1: Binding for System Application*
54 *Program Interface (API)*.
- 55 {B25} P2003.2/D8,⁶⁾ *Draft Standard for Information Technology—Test Methods*
56 *for Measuring Conformance to POSIX—Part 2: Shell and Utilities*.
- 57 {B26} JIS X0208-1990, Japanese Standards Association. *Code of the Japanese*
58 *Graphic Character Set for Information Interchange*.⁷⁾

59 3) ANSI documents can be obtained from the Sales Department, American National Standards
60 Institute, 1430 Broadway, New York, NY 10018.

61 4) CCITT documents can be obtained from the CCITT General Secretariat, International
62 Telecommunications Union, Sales Section, Place des Nations, CH-1211, Genève 20,
63 Switzerland/Suisse.

64 5) CSBS documents can be obtained from the China State Bureau of Standards, P.O. Box 8010, 42
65 Hi Chun Road, Haidian District, Beijing 100088, China.

66 6) This unapproved draft document is available from the IEEE Service Center, 445 Hoes Lane, P.O.
67 Box 1331, Piscataway, NJ 08855-1331, USA. Telephone: 1 (800) 678-IEEE or +1 (908) 981-1393
68 (outside USA).

69 7) JSA documents can be obtained from the Japanese Standards Association, 1-24, Akasaka 4-
70 chome, Minato-Ku, Tokyo, Japan 107.

- 71 {B27} JIS X0212-1990, Japanese Standards Association. *Code of the Supplemen-*
72 *tary Japanese Graphic Character Set for Information Interchange.*
- 73 {B28} KS C 5601-1987, Korean Bureau of Standards. *Korean Graphic Character*
74 *Set for Information Interchange.*⁸⁾
- 75 {B29} RFC 819, Su, Z. and Postel, J. B. *Domain naming convention for Internet*
76 *user applications.*⁹⁾
- 77 {B30} RFC 822, Crocker, D. *Standard for the format of ARPA Internet text mes-*
78 *sages.*
- 79 {B31} RFC 920, Postel, J. B. and Reynolds, J. K. *Domain requirements.*
- 80 {B32} RFC 921, Postel, J. B. *Domain name system implementation schedule —*
81 *revised.*
- 82 {B33} RFC 1123, Braden, R. T. *Requirements for Internet hosts — application and*
83 *support.*
- 84 {B34} Aho, Alfred V., Kernighan, Brian W., and Weinberger, Peter J., *The AWK*
85 *Programming Language.* Reading, MA: Addison-Wesley, 1988.
- 86 {B35} Aho, Alfred V., Sethi, Ravi, and Ullman, Jeffrey D., *Compilers, Principles,*
87 *Techniques, and Tools.* Reading, MA: Addison-Wesley, 1986.
- 88 {B36} Aho, Alfred V. and Ullman, Jeffrey D., *Principles of Compiler Design.*
89 Reading, MA: Addison-Wesley, 1977.
- 90 {B37} American Telephone and Telegraph Company, *System V Interface*
91 *Definition (SVID), Issues 2 and 3.* Morristown, NJ: UNIX Press, 1986,
92 1989.¹⁰⁾
- 93 {B38} Bolosky, Morris I. and Korn, David G., *The KornShell Command and Pro-*
94 *gramming Language.* Englewood Cliffs, NJ: Prentice Hall, 1988.
- 95 {B39} DeRemer, Frank and Pennello, Thomas J., “Efficient Computation of
96 LALR(1) Look-ahead Sets,” *SigPlan Notices*, vol. 15, no. 8, pp. 176-187, Aug.
97 1979.
- 98 {B40} Hume, A. G. and McIlroy, M. D., eds. *UNIX Programmer’s Manual.* Tenth
99 Edition. Murray Hill, NJ: AT&T Bell Laboratories, Feb. 1989.
- 100 {B41} McIlroy, M. D., ed. *UNIX Time-Sharing System Programmer’s Manual.*
101 Research Version, Eighth Edition, Volume 1. Murray Hill, NJ: AT&T Bell
102 Laboratories, Feb. 1985.

- 103 8) KBS documents can be obtained from the Korean Bureau of Standards, 2 Chung-ang-dong
104 Kwach’on-city, Kyonggi-do 171-11, Republic of Korea.
- 105 9) Internet Requests for Comments (RFC) are available from the DDN Network Information Center,
106 SRI International, Menlo Park, CA, USA 94025.
- 107 10) This is one of several documents that represent an industry specification in an area related to
108 POSIX.2. The creators of such documents may be able to identify newer versions that may be
109 interesting.

- 110 {B42} McIlroy, M. D., ed. *UNIX Time-Sharing System Programmer's Manual*.
111 Research Version, Ninth Edition, Volume 1. Murray Hill, NJ: AT&T Bell
112 Laboratories, Sep. 1986.
- 113 {B43} Ramabadrán, Tenkasi V. and Gaitonde, Sunil S., "A Tutorial on CRC Com-
114 putations," *IEEE Micro*, vol. 8, no. 4, p. 62, Aug. 1988.
- 115 {B44} Sarwate, Dilip V., "Computation of Cyclic Redundancy Checks Via Table
116 Lookup," *Communications of the ACM*, vol. 30, no. 8, p. 1011, Aug. 1988.
- 117 {B45} Knuth, Donald E., "On the Translation of Languages from Left to Right,"
118 *Information and Control*, vol. 8, no. 6, pp. 607-639, Oct. 1965.
- 119 {B46} University of California at Berkeley—Computer Science Research Group,
120 *4.3 Berkeley Software Distribution, Virtual VAX-11 Version*. Berkeley, CA:
121 The Regents of the University of California, April 1986.
- 122 {B47} /usr/group Standards Committee, *1984 /usr/group Standard*. Santa Clara,
123 CA: UniForum, 1984.
- 124 {B48} X/Open Company, Ltd., *X/Open Portability Guide, Issue 2*. Amsterdam:
125 Elsevier Science Publishers, 1987.
- 126 {B49} X/Open Company, Ltd., *X/Open Specification, X/Open Portability Guide,*
127 *Issue 3*. Reading, UK: X/Open Company, 1992.
- 128 {B50} X/Open Company, Ltd., *X/Open Specification, XPG4*. Reading, UK: X/Open
129 Company, 1992.

Annex E (informative)

Rationale and Notes

E.1 General

This informative annex is being published along with this part of ISO/IEC 9945 to assist in the process of review. It contains historical information concerning the contents of POSIX.2 and why features were included or discarded by the standard developers. It also contains notes of interest to application programmers on recommended programming practices, and to terminal users on system usage, emphasizing the consequences of some aspects of POSIX.2 that may not be immediately apparent.

Just as POSIX.2 relies on the knowledge of architecture, history, and definitions from the POSIX.1 [8], so does this annex. The reader is referred to the Rationale and Notes annex of ISO/IEC 9945-1: 1990 (IEEE Std 1003.1-1990) [8] for background material and bibliographic information about UNIX systems in general and POSIX specifically, which will not be duplicated here.

Development of POSIX.2

This portion of the general rationale outlines the organizations and documents involved in developing this part of ISO/IEC 9945.

The 1984 */usr/group Standard* [B47] originally was 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 C Standard [7], 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 IEEE P1003.2 working group was formed to address this concern. When it began work in December 1986, it limited itself to standardizing interfaces useful to portable application programs (shell scripts), as described in E.1.1. In 1988, the IEEE P1003.2 group began a parallel development effort to supplement the “base standard” for user portability. The “User Portability Extension (UPE)” was intended originally as an amendment to the base standard, to be published a year or more later. However, the balloting schedules within the IEEE and ISO/IEC JTC 1/SC22/WG15 worked out such that both documents were ready simultaneously, and this part of ISO/IEC 9945 represents a merger of these two documents.

Organization of the Rationale

Within this annex, the following terms are used:

base standard

The portions of this part of ISO/IEC 9945 addressing the original scope—application portability (most of the document outside of Section 5).

UPE The portions of this part of ISO/IEC 9945 addressing the amended scope—the User Portability Extension (all of Section 5 and a limited number of identified features in Sections 2, 3, 4, and 6).

standard developers

The individuals in the development organizations responsible for this part of ISO/IEC 9945: the IEEE P1003.2 and P1003.2a working groups, advised by the hundreds of individual technical experts who balloted the two draft standards within the IEEE Technical Committee on Operating Systems and Application Environments (TCOS), and the member bodies and technical experts of ISO/IEC JTC 1/SC22/WG15.

The remainder of this annex is organized in parallel to the normative sections and clauses of this part of ISO/IEC 9945. Rationale for the three normative annexes are contained in clauses E.8, E.9, and E.10. Within the rationale annex, references to “this clause” or “this subclause” refer to the parallel normative portion of this part of ISO/IEC 9945, unless otherwise noted.

E.1.1 Scope

This part of ISO/IEC 9945 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.”

Scope of the Base Standard

The original POSIX.2 development had the following (paraphrased) goals:

Specify a standard interface that can be accessed in common by both applications programs and user terminal-controlling programs to provide services of a more complex nature than the primitives provided by POSIX.1 {8}. This interface should be implementable on conforming POSIX.1 {8} systems. It shall include the following components:

- (1) Application program primitives to specify instructions to a command interpretation, or “shell,” facility
- (2) A standard command language for a shell that includes program execution, I/O redirection, connecting input and output data streams (with pipes), argument handling, variable substitution and expansion, and a series of control constructs similar to other high-level structured programming languages

- 75 (3) Primitives to assist applications programs and the shell language
- 76 in parsing and interpreting utility arguments
- 77 (4) A minimum set of files required for the shell and applications
- 78 (5) A group of utilities that can be called from application programs
- 79 for complex data manipulation and other tasks common to many
- 80 applications
- 81 (6) An optional group of utilities to be used for the software develop-
- 82 ment of applications
- 83 (7) Utilities and standards for the installation of applications
- 84 (8) A recommended utility syntax for utility naming and argument
- 85 specification
- 86 (9) Recommended environment variables for use by shell scripts and
- 87 application programs

88 The list of nongoals is discussed in E.1.1.2.

89 One sentence needs clarification: "This interface should be implementable on con-
90 forming POSIX.1 {8} systems." This implies that the standard developers
91 attempted to ensure that no additional functionality or extension was required to
92 implement POSIX.2 on the base defined by POSIX.1 {8}. This is not to say that
93 extensions are not allowed, but that they should not be necessary.

94 The goal "(7) Utilities and standards for the installation of applications" was once
95 interpreted to mean that an elaborate series of tools was required to install and
96 remove applications, based on complex description files and system databases of
97 capabilities. An early attempt to provide this was rejected and that type of sys-
98 tem is now being evaluated by the POSIX System Administration group for IEEE
99 P1003.7 and ISO/IEC 9945-3. However, the original goal remains in the list
100 because many of the standard utilities are, in fact, targeted specifically for appli-
101 cation installation—make, c89, lex, etc.

102 The interfaces documented for POSIX.2 are to and from high-level language appli-
103 cation programs and to and from the utilities themselves; the base POSIX.2 does
104 not directly address the interface with users.

105 The "source code" interface to the command interpreter is defined in terms of
106 high-level language functions in 7.1.1 or 7.1.2 [such as *system()*, B.3.1, or *popen()*,
107 B.3.2]. There are also other function interfaces, such as those for matching REs in
108 7.3 [*regcomp()* in B.5]. Many of the utilities in POSIX.2, and the shell itself, also
109 accept their own command languages or complex directives as input data, which
110 is also referred to as source code. This data, an ordered series of characters, may
111 be stored in files, or "scripts," that are portable between systems without true
112 recompilation. However, just as with POSIX.1 {8}, POSIX.2 addresses only the
113 issue of source code portability between systems; applications using these calls
114 may have to be recompiled or translated when moving from one system to
115 another.

116 There was considerable debate concerning the appropriate scope of the work
117 represented by this part of ISO/IEC 9945. The following alternatives were
118 evaluated:

- 119 — Define the shell and tools as extensions to POSIX.1 {8}. This would require
120 a full conforming POSIX.1 {8} system as a base for the new facilities
121 described here. Vocal proponents for this view were members of the POSIX
122 test methods working group, who foresaw difficulties in producing a
123 verification suite standard without having a known operating system base.
- 124 — Decouple the shell and tools entirely from POSIX.1 {8}. This would poten-
125 tially allow POSIX.2 to be implemented on such popular operating systems
126 as MVS/TSO, VM/CMS, MS/DOS, VMS, etc. Those systems would not have to
127 provide every minor detail of the POSIX.1 {8} language interfaces to conform
128 under this model—only enough to support the shell and tools.
- 129 — Compromise between options 1 and 2. Base POSIX.2 on an interface *similar*
130 to POSIX.1 {8}, but do not require full conformance. A simple example
131 would be a Version 7 UNIX System, which could not conform to POSIX.1 {8}
132 without considerable modification. However, a vendor could support all of
133 the features of POSIX.2 without changing its kernel or binary compatibility.
134 Another example would be a system that conformed to all stated
135 POSIX.1 {8} interfaces, but that did not have a fully conforming
136 C Standard {7} compiler. The difficulty with this option is that it made the
137 stated goal of the standard developers less precise and increased the
138 amount of analysis required for the features included.

139 The standard developers selected the third option as their goal. They chose to
140 retain the full UNIX system-like orientation, but did not wish arbitrarily to
141 deprive legitimate systems that could *almost* conform. No useful feature of shells
142 or commonly used utilities was discarded to accommodate nonconforming base
143 systems; on the other hand, no deliberate obstacles were arbitrarily erected.
144 Furthermore, POSIX.1 {8} is still required for its definitions and architectural con-
145 cepts, which are purposely not repeated in this part of ISO/IEC 9945.

146 One concrete example of how the two standards interrelate is in the usage of
147 POSIX.1 {8} function names in the descriptions of utilities in POSIX.2. There are a
148 number of historical commands that directly mapped into one of the UNIX system
149 calls. For example, `chmod` and `chmod()`; `ln` and `link()`. The standard developers
150 were faced with the problem of having to define all of the complex interactions
151 “behind the scenes” for some simple commands. Creating a file, for example,
152 involves many POSIX.1 {8} concepts, including processes, user IDs, multiple group
153 permissions (which are optional), error conditions, etc. Rather than enumerating
154 all of these interactions in many places, the standard developers chose to employ
155 the POSIX.1 {8} function descriptions, where appropriate. See the `chmod` utility in
156 4.7 as an example. The utility description includes the phrase

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

159 This means that the POSIX.2 implementor has to read the POSIX.1 {8} `chmod()`
160 description and fully understand all of its functionality, requirements, and side
161 effects, which now do not have to be repeated here. (Admittedly, this makes the
162 POSIX.2 standard a bit more difficult to read, but the standard developers con-
163 sidered that precision transcended the need for readable or semitutorial
164 documents.)

Scope of the UPE

The purpose of UPE portions of POSIX.2 is to extend the benefits of the common shell and tools environment from the application program to the interactive system user by defining optional facilities in a supplement to the original standard, a "User Portability Extension." The original base scope limited the selection of utilities and features to those useful in a noninteractive environment only. Historically, however, many of the tools described in the base standard have seen interactive use; the UPE work was predicated on the need to offer a completely consistent interactive environment for the users of such tools.

The standard developers established the following (paraphrased) goals for the UPE work:

Extend/supplement the shell services and common utilities of POSIX.2 for POSIX.1 {8} and similar systems. Provide extensions to those services and additional utilities to support terminal users in a consistent manner across all conforming systems. The extension is limited by three constraints:

- (1) The *users* in this context are limited to the group of individuals who are familiar with the style of interaction characteristic of historically derived systems based on one of the UNIX operating systems. Typical users would include program developers, engineers, or general-purpose time-sharing users.
- (2) The *environment* to be supported is a multi-user time-sharing system supporting character-oriented display terminals. Alternatively, it is a collection of single-user systems interconnected via local area networks or telephone lines, but with similar user interfaces. POSIX.2 does not include support tailored for bit-mapped or graphics display terminals, although it is expected that such terminals could emulate the character orientation required by this environment. When facilities require cursor addressability from the terminal hardware, these shall be specifically identified.
- (3) The facilities to be provided are based on the historical models of the following documents: the *System V Interface Definition* {B37}; the *BSD User Manual* {B46}; the *X/Open Portability Guide* {B49}; and, documentation for the KornShell {B38}. Emphasis will be placed on standardizing existing practice for existing users, with changes or additions limited to correcting deficiencies in the following areas:
 - (a) Support for international character sets and other localization requirements, such as date formats, collation sequences, etc.
 - (b) Reconciliation of differences between the historical implementations.
 - (c) Elimination of system or device dependencies.
 - (d) Corrections of features that could reduce system or user security/integrity.

- 210 The extended facilities shall be designed to be cognizant of international
211 character sets and other localization requirements, such as date formats,
212 collation sequences, etc.
- 213 The extended facilities to be provided may include
- 214 — Enhancements to utilities already defined in the base portion of
215 POSIX.2 to facilitate interactive use.
 - 216 — Utilities for editing and displaying files on full-screen character-
217 oriented display terminals.
 - 218 — Utilities for querying and managing the environment of the user.
 - 219 — Utilities for interacting with other users on the system or local
220 network.
 - 221 — Utilities for scheduling and controlling the background, or periodi-
222 cally running, work of the user.
 - 223 — Utilities to enhance the interactive program development activities,
224 complementary to those tools used for software installation in the
225 original project.
 - 226 — Other user portability utilities and features within the general limi-
227 tations described by this scope.
- 228 The scope was carefully worded to exclude the development of interfaces for new
229 technology environments, such as bit-mapped graphics windowing terminals and
230 “look-and-feel” considerations (icons, menus, mice, etc.) for novice or specialized
231 system users. There are numerous other standards groups in this domain and
232 POSIX.2 did not want to overlap with them, but merely satisfy historical users.
- 233 There was periodic debate among the standard developers about how some of the
234 interactive standard utilities could be implemented, given the set of features pro-
235 vided by POSIX.1 {8}, the presumed base operating system. For example, the
236 *nice* utility is not necessarily supported by a *nice()* system call or function
237 because one does not exist in POSIX.1 {8}. Thus, *nice* cannot be a Strictly Con-
238 forming POSIX Shell Application; at most it could be a Conforming POSIX Shell
239 Application with Extensions. However, there is no requirement stated in POSIX.2
240 that the standard utilities have to be strictly conforming programs themselves.
241 The original charter for developing the base standard expressed a goal that “This
242 interface should be implementable on conforming POSIX.1 {8} systems.” Most, if
243 not all, of the standard utilities in the base standard met that goal, but the
244 expanded scope for the UPE made that more difficult to achieve. A few of the util-
245 ities have some requirements for administrative support that were outside the
246 scope of POSIX.1 {8}: *crontab*, *df*, *nice*, *ps*, *renice*, and *who* are examples. It
247 is anticipated that the emerging POSIX.7 System Administration work will pro-
248 vide standard underpinnings for such services, but until they are approved, users
249 of this part of ISO/IEC 9945 will have to rely upon common historical implementa-
250 tion extensions.
- 251 There were a number of differences in philosophy used in preparing the utilities
252 in the UPE from those in the base standard:

- 253 — It was assumed that a human was the audience for the output of utilities.
254 Unlike a shell script, the primary audience of the base standard, the
255 human is not so concerned about exactness of format. Therefore, in many
256 cases, the UPE merely suggests the form of the output, rather than using
257 the *printf()*-like form of the base standard. This has the benefit of allowing
258 multiple historical implementations to conform without change. There are
259 exceptions to this general philosophy, however. In some cases, such as *df*,
260 the standard developers considered that users were confused by the historical
261 differences in output. A compromise “portable” output format that
262 could be provided by all systems in an upward-compatible manner was pro-
263 vided. That this output could then be parsed by a shell script is a serendi-
264 pitous side effect. (As explained in the original base standard, *df* is a util-
265 ity that produces output requiring human judgment for its interpretation;
266 shell scripts should probably avoid using *df* to make decisions about allo-
267 cating new files.)
- 268 — Since the idea of the UPE was to form a common environment for system
269 users of existing technology terminals, and not to lead the industry in
270 developing new user interfaces, the standard developers chose to limit
271 changes to command names and syntax. While the application orientation
272 of the base standard caused various name and option changes for con-
273 sistency and historical harmonization, the standard developers chose to
274 leave many of the unusual idiosyncrasies intact so that users would con-
275 tinue to feel comfortable at their terminals. It is for this reason that the
276 UPE has comparatively fewer obsolescent features than the base standard.

277 E.1.1.1 Historical Practice

278 The standard developers could have developed a standard that allowed all histori-
279 cal implementations (i.e., those existing prior to the time of publication) to be fully
280 conforming and all historical applications to be Strictly Conforming POSIX.2
281 Applications without requiring any changes. Some modifications will be required
282 to reconcile the specific differences between historical implementations; there are
283 many divergent versions of UNIX systems extant, and applications sometimes
284 have been written to take advantage of features (or bugs) on specific systems.
285 Therefore, the standard developers established a set of goals to maximize the
286 value of the POSIX.2 it eventually produced. These goals are enumerated in the
287 following four subclauses. They are listed in approximate priority sequence,
288 where the first subclause is the most important portability goal.

289 E.1.1.1.1 Preserve Historical Applications

290 The most important priority was to ensure that historical applications continued
291 to operate on conforming implementations. This required the selection of many
292 utilities and features from the most prevalent historical implementations. The
293 standard developers relied on the following factors:

- 294 (1) Many inconsistent historical features will still be supported as
295 *obsolescent*.

- 296 (2) Common features of System V and BSD will continue to be supported by
297 their sponsors, even if they are not included here (just as long as they are
298 not prevented from existing).

299 Therefore, POSIX.2 was written so that the large majority of well-written histori-
300 cal applications could continue to operate as Conforming POSIX.2 Applications
301 Using Extensions.

302 **E.1.1.1.2 Improve the Interfaces**

303 The standard developers chose to extend the benefits of historical UNIX systems
304 by making limited improvements to the utility interfaces; numerous complaints
305 have been made about the inconsistencies in the command-line interface, which
306 have allegedly made it harder for novice users. Given the constraints of E.1.1.1.1,
307 the standard developers made the following general modifications:

- 308 (1) Utilities were extended to deal with differences in character sets, collat-
309 ing sequences, and some cultural aspects relating to the locale of the
310 user. (Examples: new features in REs; new formatting options in date;
311 see 4.15.)

- 312 (2) The Utility Syntax Guidelines in 2.10.2 were applied to almost all of the
313 utilities to promote a consistent interface. The guidelines themselves are
314 less constrained than their counterparts in the SVID [B37].

315 In many cases, historical utilities have not conformed with these guide-
316 lines (which were written considerably later than the utilities them-
317 selves). The older interfaces have been maintained in POSIX.2 as obsoles-
318 cent features. (Examples: `join`, `sort`.) However, in some cases, such
319 as `dd`, `find`, and `pr`, such major revision was required that the standard
320 developers decided to leave the historical interfaces as is. “Fixing” the
321 interface would mean replacing the command, which would not help
322 applications portability. So, fixing was limited to relatively minor excep-
323 tions to the new guidelines, where reasonable consistency could be
324 achieved while still maintaining the general type of interface of the his-
325 torical version.

- 326 (3) Features that were not generally portable across machine architectures
327 or systems were omitted or marked obsolescent. New, more portable
328 interfaces were introduced. (Examples: the octal number methods of
329 describing file modes in `chmod` and other utilities were marked obsoles-
330 cent; the symbolic “`ugo`” method has been extended to other utilities,
331 such as `umask`.)

- 332 (4) Features that have proved to be popular in some specific UNIX system
333 variants were adopted. (Examples: `diff -c`, which originated in BSD
334 systems, and the “new” `awk` from System V.) Such features were selected
335 given the requirements for balloting group consensus; the features had to
336 be used widely enough to balance accusations of “creeping featurism” and
337 violations of the UNIX system “tools philosophy.”

- 338 (5) Unreasonable inconsistencies between otherwise similar interfaces were
339 reconciled. (Example: methods of specifying the patterns to the three

- 340 grep-related utilities have been made more consistent in single grep in
341 POSIX.2.)
- 342 (6) When irreconcilable differences arose between versions of historical utili-
343 ties, new interfaces (utility names or syntax) were sometimes added in
344 their places. The standard developers resisted deviating significantly
345 from historical practice; the new interfaces are generally consistent with
346 the philosophy of historical systems and represent comparable func-
347 tionality to the interfaces being replaced. In some cases, System V and
348 BSD had diverged (such as with `echo` and `sum`) so significantly that no
349 compromises for a common interface were possible. In these cases, either
350 the divergent features were omitted or an entirely new command name
351 was selected (such as with `printf` and `cksum`).
- 352 (7) Arbitrary limits to utility operations were omitted. (Example: some his-
353 torical `ed` utilities have very limited capabilities for dealing with large
354 files or long input lines.)
- 355 (8) Arbitrary limitations on historical extensions were eliminated. (Exam-
356 ple: REs have been described so that the popular `< ... >` extension is
357 allowed.)
- 358 (9) Input and output formats were specified in more detail than historical
359 implementations have required, allowing applications to operate more
360 effectively in pipelines with these utilities. (Example: `comm`.)

361 Thus, in many cases the standard developers could be accused of “violating histor-
362 ical practice,” and in fact received some objections to that effect from implemen-
363 tors (although rarely from users or application developers). The standard
364 developers were sensitive to charges that they were engaged in arbitrary software
365 engineering rather than merely codifying existing practice. When changes were
366 made, they were always written to preserve historical applications, but to move
367 new conforming applications into a more consistent, portable environment. This
368 strategy obviously required changes to historical implementations; the standard
369 developers carefully evaluated each change, weighing the value to users against
370 the one-time costs of adding the new interfaces (and of possibly breaking applica-
371 tions that took advantage of bugs), generally siding with the users when the costs
372 to implementations and applications was not excessively high.

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

376 E.1.1.1.3 Allow Historical Conforming Applications

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

384 **E.1.1.1.4 Preserve Historical Implementations**

385 As explained in E.1.1.1.2, the requirements for portability and a consistent inter-
386 face have caused the standard developers to add new utilities and features. No
387 historical implementations contained all of the attributes required by the stan-
388 dard developers. Therefore, this lowest priority goal was affected by the preced-
389 ing goals, and every known historical implementation will require some
390 modifications to conform to POSIX.2.

391 The standard developers took care to ensure that the implementations could add
392 the new or modified features without breaking the operation of existing applica-
393 tions. (Note that the standard utilities are not considered applications in this
394 regard, but are part of the implementation. In fact, many or most of the utilities
395 named by POSIX.2 will have to change to some extent.)

396 **E.1.1.2 Outside the Scope**

397 The following areas are outside the scope of this part of ISO/IEC 9945. This sub-
398 clause explains more of the rationale behind the exclusions.

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

401 The standard developers followed the lead of the POSIX.1 group in this
402 instance. Administrative commands were considered to be too implemen-
403 tation dependent and not useful for application portability. Subsequent
404 to this decision, a separate POSIX.7 system administration working group
405 was formed to deal with this area of “operator portability.” It is antici-
406 pated that utilities needed for system administration will be closely coor-
407 dinated with the POSIX.2 standard developers.

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

410 This area is similar to item (1). System installation is contrasted against
411 the application installation portion of the Scope by its orientation to ins-
412 talling the operating system itself, versus application programs. The
413 exclusion of operating system installation facilities should not be inter-
414 preted to mean that some of standard utilities *cannot* be used for instal-
415 ling operating system components.

- 416 (3) *Networking commands*

417 These were excluded because they are deeply involved with other
418 standards-making bodies. In this case, several working groups were
419 formed within the POSIX family to deal with this area. It is anticipated
420 that utilities needed for networking, if any, will be closely coordinated
421 with the POSIX.2 standard developers. [In early drafts of this part of
422 ISO/IEC 9945, which predated the formation of the networking-specific
423 POSIX working groups, the historical “UNIX system to UNIX system copy
424 (UUCP)” programs and protocols were included. These descriptions have
425 been omitted 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.)*

Until the approval of the UPE project, this was probably the most contentious exclusion. A common complaint about many historical UNIX systems was how they were not very “user friendly.” Some people had hoped that the interface to users could be standardized with mice, icon-based desktop metaphors, and so forth. The standard developers avoided these concerns by reminding their audience that POSIX.2 was an application portability standard and therefore had 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 to assume that a user can interact with an application. That is why such facilities as displaying strings (with `printf`) without `<newline>`s, `stty`, and various prompting utilities are included in this part of ISO/IEC 9945.

This nongoal is not applicable to the UPE portions of POSIX.2.

- (5) *Graphics programs or interfaces*

See the comments on user interface in (4). More graphically oriented user interfaces remain outside of the scope because there are active standards efforts underway in other committees.

- (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 responsibility of other standards bodies.

E.1.1.3 Language-Independent Descriptions

The POSIX.1 working group is 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 become one of several language bindings that interface with the core services descriptions. The POSIX.2 standard developers did not wish to duplicate effort and have therefore waited until POSIX.1 [8] achieves progress in this area. Fortunately, POSIX.2 is substantially less oriented toward C than POSIX.1 [8] is. In fact, all of the C interfaces are entirely optional.

E.1.1.4 Base Documents

The standard developers consulted a number of documents in the course of their deliberations, to select utilities and features. There were five primary documents that started off the developments:

- 467 (1) *The System V Interface Definition* (SVID), Issue 2, Volume 2 {B37}.
- 468 (2) *The X/Open Portability Guide* (XPG), Issues II {B48} and III {B49},
469 Volume 1.
- 470 (3) *The UNIX User's Reference Manual*, 4.3 Berkeley Software Distribution,
471 Virtual VAX-11 Version {B46}. (The printed documentation as well as the
472 online versions provided with the BSD "Tahoe" and "Reno" distributions
473 were considered as one base document for the POSIX.2 work.)
- 474 (4) *The KornShell Command and Programming Language* {B38}.
- 475 (5) *The AWK Programming Language* {B34}.

476 The XPG {B49} was used primarily in initial deliberations about which utilities
477 and features to include. The X/Open companies had done a very thorough job in
478 analyzing the SVID {B37} and other standards to compile a list of the most useful
479 and portable utilities. They carefully marked many features that had portability
480 problems, and the standard developers avoided them for POSIX.2.

481 AT&T, X/Open, and Berkeley provided machine-readable documentation for the use
482 of the standard developers. However, due to substantial differences in formatting
483 standards, there is little resemblance between some of the utilities described here
484 and their counterparts in the SVID {B37}, XPG {B49}, and BSD user manual {B46}.
485 Nevertheless, early usage of these documents was an invaluable aid in the produc-
486 tion of this part of ISO/IEC 9945, and the POSIX.2 standard developers extend their
487 sincere thanks to all three organizations for their generous cooperation.

488 The biggest divergence in the documentation style adopted by POSIX.2 has been its
489 philosophy of fully specifying interfaces. The SVID {B37} and XPG {B49} are
490 oriented solely towards application portability. Implementors would have a
491 difficult time writing some of these utilities from the descriptions alone. In fact,
492 both documents freely rely on the potential implementors licensing the source code
493 for the reference systems to complete the specification. POSIX.2, on the other hand,
494 also has implementors in its audience, and the part of ISO/IEC 9945 expanded its
495 descriptions wherever useful and feasible. For example, it makes use of yacc-like
496 grammars to describe complex syntaxes. It attempts to describe the interactions
497 between options, operands, and environment variables, where conflicts can exist.
498 It also attempts to describe all of the useful utility input and output formats. The
499 goal was to allow application developers to write filters or other programs that
500 could parse the output of any of these utilities or to provide meaningful input from
501 their programs. To the knowledge of the standard developers, this is a task never
502 before attempted for the historical UNIX system commands—the source code was
503 always readily available to anyone who needed to know this information.

504 The two commercial books listed were used as reference materials in preparing
505 information on the shell and the *awk* language that was more recent and com-
506 plete than documentation available from AT&T or X/Open at that time.

507 Selecting the KornShell documentation {B38} was a clear signal from the stan-
508 dard developers that *ksh* extended scripting features and interactive command-
509 line editing features were major factors in selecting POSIX shell behavior. The
510 System V shell and the C-shell are also in the base documents, but numerous dis-

511 discussions among the standard developers resulted in the following guidelines used
512 to prepare and ballot the drafts:

- 513 — Use the System V shell as the broad shell scripting base, but select Korn-
514 Shell features to correct historical deficiencies (such as the lack of easy
515 arithmetic and substring manipulation, the tilde expansion, and the prob-
516 lems of command substitution nesting).
- 517 — Use the KornShell model for interactive features such as command-line
518 editing.
- 519 — When required features appeared in both the C-shell and KornShell,
520 emphasize the KornShell interface unless it was clearly deficient. This
521 decision was predicated on the fact that it made less sense to select script-
522 ing features and interactive features from two very different shells. Histor-
523 ical C-shell users would find the POSIX shell to be rather different than
524 their experience, and selecting isolated C-shell features would only serve to
525 confuse both C-shell and KornShell users.

526 E.1.1.5 Organization of the Document

527 The standard document is organized into sections. Some of these, such as the
528 Scope in 1.1, are mandated by ISO/IEC, the IEEE, and other standards bodies.
529 The remainder of the document is organized into (comparatively) small sections
530 for the convenience of the standard developers and others. It has been suggested
531 that all of the utility descriptions (and maybe the functions, too) should be organ-
532 ized into one large section, all in alphabetical order. This would presumably
533 make it easier for some users to use the document as a reference document. The
534 standard developers deliberately chose to not organize it in this way for the fol-
535 lowing reasons:

- 536 (1) Certain sections are optional. It is more convenient for the internal
537 references within this part of ISO/IEC 9945, and also for people specifying
538 systems, if these optional sections are in large pieces, rather than a
539 detailed list of utility names.
- 540 (2) Future amendments to POSIX.2 will be adding new utilities that will also
541 be optional. It would be confusing to try to merge documents at a level
542 below major sections (chapters).

543 E.1.2 Normative References

544 There is no additional rationale provided for this subclause.

545 E.1.3 Conformance

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

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

549 The descriptions of the ISO/IEC and <National Body> Conforming POSIX.2 Appli-
550 cations are similar to the same descriptions in POSIX.1 {8}. This is not a duplica-
551 tion of effort, as POSIX.2 relies on only a portion of POSIX.1 {8}, as explained in 1.1
552 and 2.9.1. Therefore, conformance to POSIX.2 has to be described separately from
553 any conformance options or requirements in POSIX.1 {8}.

554 There is no reference to the Language-Independent System Services Option in the
555 list of optional features that may be provided by the conforming implementation.
556 There is no conformance value provided by that section, except as a reference
557 point for functions actually provided by a real language binding. Therefore, the
558 language binding sections are the ones that remain in the optional list.

559 The documentation requirement for implementation extensions (“shall define an
560 execution environment”) is simply meant to require that system-wide or per-user
561 configuration options or environment variables that affect the operation of appli-
562 cations that use the standard utilities and functions be described in the confor-
563 mance document. For example, if setting the (imaginary) **LC_TRUTH** variable
564 causes changes in the exit status of true, the conformance document must
565 describe this condition and how to avoid it—for example, by unsetting the vari-
566 able in the login script (see the unset utility in 3.14.14).

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

569 **E.1.4 Test Methods**

570 The IEEE P2003.2 {B25} test methods standard (originally designated P1003.3.2)
571 was started considerably after the original base standard and will not be com-
572 pleted before POSIX.2 itself. Thus, there is text indicating that references to it are
573 only for informative, bibliographic purposes. However, it is the intent of the stan-
574 dard developers (who participated in the test methods development) that these
575 test methods eventually be used to determine implementation conformance to
576 POSIX.2 (within the limits established by the POSIX.3 series of standards—see
577 POSIX.3 {B23}). (Test methods cannot ensure conformance by themselves because
578 there are requirements placed on a conforming implementation that cannot be
579 verified by test procedures). This will occur when the test methods are at a com-
580 parable level of acceptance. For example, when POSIX.2 is an IEEE standard, the
581 test methods will be sanctioned officially by the IEEE when P2003.2 {B25} is an
582 IEEE standard. When POSIX.2 becomes an International Standard, the test
583 methods will be acknowledged by JTC 1 when the test methods also become an
584 International Standard.

585 It should be recognized that the test methods, described in a more recent docu-
586 ment, may deviate slightly from the original POSIX.2 because they can take
587 advantage of the official interpretations process that occurs after POSIX.2 is pub-
588 lished. The two standards should be used together to describe the POSIX Shell
589 and Utilities environment, particularly when additional specificity or clarity can
590 be obtained from P2003.2 {B25}, based on the interpretation process.

591 E.2 Terminology and General Requirements

592 E.2.1 Conventions

593 The C language was chosen for many examples because

- 594 — It eliminates any requirement to document a different pseudo-code
- 595 — It is a familiar language to many of the potential readers of POSIX.2
- 596 — It is the language most widely used for historical implementations of the
- 597 utilities

598 E.2.2 Definitions

599 E.2.2.1 Terminology

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

602 The reader is referred to the definition of *program* in 2.2.2.140 to understand the
603 expression “program construction.” The use of *program* in POSIX.2 is differen-
604 tiated from the emphasis in POSIX.1 {8} only on high-level languages by the
605 broader concern in POSIX.2 with utility and command-language interactions.
606 Included in the scope of program construction are

- 607 (1) Shell command language
- 608 (2) Command arguments
- 609 (3) REs of various types
- 610 (4) Command input language syntax, such as *awk*, *bc*, *ed*, *lex*, *make*, *sed*,
611 and *yacc*. Some of these are so complex that they rival traditional high-
612 level languages.

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

615 The term *obsolescent* means “do not use this feature in new applications.” The
616 obsolescence concept is not an ideal solution, but was used as a method of increas-
617 ing consensus: many more objections would be heard from the user community if
618 some of these historical features were suddenly withdrawn without the grace
619 period obsolescence implies. The phrase “may be considered for withdrawal in
620 future revisions” implies that the result of that consideration might in fact keep
621 those features indefinitely if the predominance of applications do not migrate
622 away from them quickly.

623 The term *application* is not defined in either POSIX.1 {8} or POSIX.2; it is assumed
624 to be a part of general computer science terminology. The definition in this sub-
625 clause is present to ensure readers understand that the general requirements for
626 applications also apply to interactive users. If, for instance, POSIX.2 requires a
627 Strictly Conforming POSIX.2 Application to use a specific option to a utility, an

628 interactive user would also have to use that option to be considered strictly
629 conforming.

630 **E.2.2.2 General Terms**

631 Many of the terms originated in POSIX.1 [8] and are duplicated in POSIX.2 to meet
632 editorial requirements. In some cases, there is supplementary text that presents
633 additional information concerning POSIX.2 aspects of the concept.

634 There are some terms used that are undefined in POSIX.2, POSIX.1 [8], or the
635 C Standard [7]. The standard developers believed that these terms have a “com-
636 mon usage” and that a definition in POSIX.2 would not be appropriate. Terms in
637 this category include, but are not limited to, the following: *application*, *character*
638 *set*, *cursor*, *login session*, *user*. Good sources for general terms of this type are the
639 *ISO/AFNOR Dictionary of Computer Science* [B16] and *The New IEEE Dictionary*
640 *of Electrical and Electronics Terms* [B20].

641 **Base Definitions**

642 POSIX.2 uses the term *character* to mean a sequence of one or more bytes
643 representing a single graphic symbol, as defined in POSIX.1 [8]. The deviation in
644 the exact text of the C Standard [7] definition for *byte* meets the intent of the
645 Rationale of the C Standard [7] and of the developers of POSIX.1 [8], but it also
646 clears up the ambiguity raised by the term *basic execution character set*, which is
647 not defined in POSIX.1 [8]. It is expected that a future version of POSIX.1 [8] will
648 align with the text used here. The octet-minimum requirement is merely a
649 reflection of the {CHAR_BIT} value in POSIX.1 [8] and the C Standard [7].

650 The POSIX.1 [8] term *file mode* is a superset of the POSIX.2 *file mode bits*.
651 POSIX.1 [8] defines the file mode as the entire *mode_t* object (which includes the
652 file type in the upper four bits historically, the sticky bit on most implementa-
653 tions, and potentially other nonstandardized attributes), while POSIX.2 file mode
654 bits include only the 11 defined bits.

655 The terms *command* and *utility* are related but have distinct meanings. Com-
656 mand is defined as “a directive to a shell to perform a specific task.” The directive
657 can be in the form of a single utility name (for example, *ls*), or the directive can
658 take the form of a compound command (for example, *ls | grep name | pr*). A
659 utility is a program that can be called by name from a shell. Issuing only the
660 name of the utility to a shell is the equivalent of a one-word command. A utility
661 may be invoked as a separate program that executes in a different process than
662 the command language interpreter, or it may be implemented as a part of the
663 command language interpreter. For example, the *echo* command (the directive to
664 perform a specific task) may be implemented such that the *echo* utility (the logic
665 that performs the task of echoing) is in a separate program; therefore, it is exe-
666 cuted in a process that is different than the command language interpreter. Con-
667 versely, the logic that performs the *echo* utility could be built into the command
668 language interpreter; therefore, it could execute in the same process as the com-
669 mand language interpreter.

670 The terms *tool* and *application* can be thought of as being synonymous with *utility*
671 from the perspective of the operating system kernel. Tools, applications, and util-
672 ities historically have run, typically, in processes above the kernel level. Tools

and utilities historically have been a part of the operating system nonkernel code and have 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 they 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 distributor, applications by third-party software distributors or by the users themselves. Nevertheless, POSIX.2 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 could not 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 part of ISO/IEC 9945 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 or they explicitly describe their limitations within their individual subclauses. The definition of *text file* has caused 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 POSIX.2 is not a 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 part of ISO/IEC 9945.

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. A future version of POSIX.2 is expected to address symbolic link issues; see H.4.

The term *file name* was defined in early 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}. The standard developers considered that maintaining ties to the base documents was an important consideration for the proper review of this part of ISO/IEC 9945. A summary of the name differences appears in E.2.4. 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 POSIX.2 attempts to balance readability against precision.

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

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

737 In most 1 b character sets, such as ASCII, the concept of column positions is ident-
738 ical to character positions and to bytes. Therefore, it has been historically accept-
739 able for some implementations to describe line folding or tab stops or table
740 column alignment in terms of bytes or character positions. Other character sets
741 pose complications, as they can have internal representations longer than one
742 octet and they can have display characters that have different widths on the ter-
743 minal screen or printer.

744 In POSIX.2 the term *column positions* has been defined to mean character—not
745 byte—positions in input files (such as “column position 7 of the FORTRAN input”).
746 Output files describe the column position in terms of the display width of the nar-
747 rowest printable character in the character set, adjusted to fit the characteristics
748 of the output device. It is very possible that *n* column positions will not be able to
749 hold *n* characters in some character sets, unless all of those characters are of the
750 narrowest width. It is assumed that the implementation is aware of the width of
751 the various characters, deriving this information from the value of `LC_CTYPE`,
752 and thus can determine how many column positions to allot for each character in
753 those utilities where it is important. This information is not available to the port-
754 able application writer because POSIX.2 provides no interface specification to
755 retrieve such information. Support for such interfaces will be evaluated for a
756 future version of POSIX.2. These may include modifications to the charmap for-
757 mat to define the widths of character (or groups of characters) and a function to
758 return the information for a particular character or string. See H.1.

759 The term *column position* was used instead of the more natural *column* because
760 the latter is frequently used in POSIX.2 in the different contexts of columns of
761 figures, columns of table values, etc. Wherever confusion might result, these
762 latter types of columns are referred to as *text columns*.

763 The ISO/IEC 646 {1} character set standard permits substitution of the “currency
764 symbol” for the character \$ in the “reference character set” (which is the same as
765 ASCII), as well as the pound sign for the number sign. An early draft of POSIX.2
766 explicitly permitted the substitution of those characters. However, at the request
767 of the WG15 POSIX working group, this dependency on a specific character encod-
768 ing standard was removed. POSIX.2 uses the ASCII-oriented names and their

769 symbols, but it is valid for an implementation using ISO/IEC 646 {1} to accept, for
770 instance, the pound sign (£) as a comment character in the shell, if that is what
771 the character set of the locale uses instead of the number sign (#). Since these
772 character pairs use the same encoded values, it would not be likely that could be
773 perceived by a test suite.

774 The term *stream* is not related to the System V STREAMS communications facil-
775 ity; it is derived from historical UNIX system usage and has been made official by
776 the C Standard {7}. The POSIX.2 standard makes no differentiation between the
777 *text stream* and *binary stream* of the C language.

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

780 **UPE Definitions**

781 POSIX.2 assigns precise requirements for the terms *display* and *write*, unlike
782 some historical user manuals. Some historical systems have chosen to implement
783 certain utilities without using the historical UNIX system file descriptor model.
784 For example, the *vi* editor might employ direct screen memory updates on a per-
785 sonal computer, rather than a *write()* system call. An instance of user prompting
786 might appear in a “pop-up” dialog box, rather than with standard error. When
787 POSIX.2 uses the term *display*, the method of outputting to the terminal is
788 unspecified; many historical implementations use *termcap* or *terminfo*, but this is
789 not a requirement. The term *write* is used when POSIX.2 mandates that a file
790 descriptor be used and that the output can be redirected. However, it is assumed
791 that when the writing is directly to the terminal (it has not been redirected else-
792 where), there is no practical way for a user or test suite to determine whether a
793 file descriptor is actually being used or not. Therefore, the use of a file descriptor
794 is mandated only for the redirection case, and the implementation is free to use
795 any method when the output is not redirected. Early drafts included the terms
796 *print* and *type*, which attempted to exert a finer degree of specification on output
797 and input differences for terminals versus other types of files. These verbs were
798 discarded when it was realized that there is no practical or useful way to deter-
799 mine how I/O is accomplished when it is not redirected. *Read* and *write* are used
800 almost exclusively, with the very few exceptions of those utilities where output
801 redirection need not be supported: *tabs*, *talk*, *tput*, and *vi*.

802 The definition of *block-mode terminal* refers to a mode of operation because some
803 terminals are capable of full character-at-a-time operation only in certain
804 hardware- or software-selected modes. Terminals that can support all the
805 features of POSIX.2 in one mode should not be excluded if there are some cases
806 where they may be set up with other characteristics. There is no implication that
807 the implementation is required to sense changes in these modes during operation
808 of the utilities.

809 The definition of *scroll* omits the situation where the cursor moves while the data
810 remains stationary; this is considered to be a cursor movement, not scrolling.

811 The definition of *system scheduling priority* is not intended to suggest that all
812 processes in a system have priorities that are comparable. Scheduling policy
813 extensions such as the realtime priorities in POSIX.4 make the notion of a single
814 underlying priority for all scheduling policies problematic. Some systems may

implement the `nice`-related features to affect all processes on the system, others to affect just the general time-sharing activities implied by POSIX.2, and others may have no effect at all. Because of the use of “implementation defined” in `nice` and `renice`, a wide range of implementation strategies are possible.

Shell Definitions

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 `$(` arithmetic form]. The decision to require this was 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 historical scripts is even smaller.

There are no explicit limits in POSIX.2 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.

The characters allowed in alias names are from historical practice.

E.2.2.3 Abbreviations

There is no additional rationale provided for this subclause.

E.2.3 Built-In Utilities

In early 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 need the capability 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.

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

858 (1) This would require wasting space in the file system, at the expense of
859 very small systems. However, it has been pointed out that all 16 utilities
860 in the table can be provided with 16 links to a single-line shell script:

861 `$0 "$@"`

862 (2) It is not logical to require invocation of utilities like *cd* because they have
863 no value outside the shell environment or cannot be useful in a child pro-
864 cess. However, counter-examples always seemed to be available for even
865 the most unusual cases:

866 `find . -type d -exec cd {} ; -exec foo {} ;`
867 (which invokes *foo* on accessible directories)

868 `ps ... | sed ... | xargs kill`

869 `find . -exec true ; -a ...`
870 (where *true* is used for temporary debugging)

871 (3) It is confusing to have a utility such as *kill* that can easily be in the file
872 system in the base standard, but that requires built-in status for the UPE
873 (for the % job control job ID notation). It was decided that it was more
874 appropriate to describe the required functionality (rather than the imple-
875 mentation) to the system implementors and let them decide how to
876 satisfy it.

877 On the other hand, it was realized that any distinction like this between utilities
878 was not useful to applications and that the cost to correct it was small. These
879 arguments were ultimately the most effective.

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

881 *alias*, *fc*, *unalias*

882 The functionality of these utilities is performed more simply
883 within the shell itself and that is the model most historical imple-
884 mentations have used.

885 *bg*, *fg*, *jobs*

886 All of the job-control related utilities are eligible for built-in
887 status because that is the model most historical implementations
888 have used.

889 *cd*, *getopts*, *newgrp*, *read*, *umask*, *wait*

890 The functionality of these utilities is performed more simply
891 within the context of the current process. An example can be
892 taken from the usage of the *cd* utility. The purpose of the utility
893 is to change the working directory for subsequent operations. The
894 actions of *cd* affect the process in which *cd* is executed and all
895 subsequent child processes of that process. Based on the
896 POSIX.1 [8] process model, changes in the process environment of
897 a child process have no effect on the parent process. If the *cd*
898 utility were executed from a child process, the working directory
899 change would be effective only in the child process. Child
900 processes initiated subsequent to the child process that executed

901 the `cd` utility would not have a changed working directory rela-
902 tive to the parent process.

903 `command` This utility was placed in the table primarily to protect scripts
904 that are concerned about their **PATH** being manipulated. The
905 “secure” shell script example in E.4.12 would not be possible if a
906 **PATH** change retrieved an alien version of `command`. (An alter-
907 native would have been to implement `getconf` as a built-in, but
908 the standard developers considered that it carried too many
909 changing configuration strings to require in the shell.)

910 `kill` Since `kill` provides optional job control functionality using shell
911 notation (`%1`, `%2`, etc.), some implementations would find it
912 extremely difficult to provide this outside the shell.

913 `true, false` These are in the table as a courtesy to programmers who wish to
914 use the “while `true`” shell construct without protecting `true`
915 from **PATH** searches. (It is acknowledged that “while `:`” also
916 works, but the idiom with `true` is historically pervasive.)
917

918 All utilities, including those in the table, are accessible via the functions in 7.1.1
919 or 7.1.2 [such as `system()` or `popen()`]. There are situations where the return func-
920 tionality of `system()` and `popen()` is not desirable. Applications that require the
921 exit status of the invoked utility will not be able to use `system()` or `popen()`, since
922 the exit status returned is that of the command language interpreter rather than
923 that of the invoked utility. The alternative for such applications is the use of the
924 `exec` family. (The text concerning conformance to POSIX.1 {8} was included
925 because where `exec` is not provided in the underlying system, there is no way to
926 require that utilities be `exec`-able.)

927 E.2.4 Character Set

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

932 POSIX.2 poses no requirement that multiple character sets or code sets be sup-
933 ported, leaving this as a marketing differentiation for implementors. Although
934 multiple *charmap* files are supported, it is the responsibility of the implementa-
935 tion to provide the file(s); if only one is provided, only that one will be accessible
936 using the `localedef -f` option (although in the case of just one file on the sys-
937 tem, `-f` is not useful).

938 The statement about invariance in code sets for the portable character set is
939 worded to avoid precluding implementations where multiple incompatible code
940 sets are available (for instance, ASCII and EBCDIC). The standard utilities cannot
941 be expected to produce predictable results if they access portable characters that
942 vary on the same implementation.

943 Not all character sets need include the portable character set, but each locale
944 must include it. For example, a Japanese-based locale might be supported by a

945 mixture of character sets: JIS X 0201 Roman (a Japanese version of ISO/IEC 646),
946 JIS X 0208, and JIS X 0201 Katakana. Not all of these character sets include the
947 portable characters, but at least one does (JIS X 0201 Roman).

948 Encoding mechanisms based on single shifts, such as the EUC encoding used in
949 some Asian and other countries, can be supported via the current charmap
950 mechanism. With single-shift encoding, each character is preceded by a shift code
951 (SS2 or SS3). A complete EUC code, consisting of the portable character set (G0)
952 and up to three additional character sets (G1, G2, G3), can be described using the
953 current charmap mechanism; the encoding for each character in additional char-
954 acter sets G2 and G3 must then include their single-shift code. Other mechan-
955 isms to support locales based on encoding mechanisms such as locking shift will
956 also be considered for a future version of POSIX.2.

957 The character set description file provides

- 958 — The capability to describe character set attributes (such as collation order
959 or character classes) independent of character set encoding, and using only
960 the characters in the portable character set. This makes it possible to
961 create “generic” localedef source files for all code sets that share the
962 portable character set (such as the ISO 8859 family or IBM Extended ASCII).
- 963 — Standardized symbolic names for all characters in the portable character
964 set, making it possible to refer to any such character regardless of
965 encoding.

966 Implementations are free to choose their own symbolic names, as long as the
967 names identified by this part of ISO/IEC 9945 are also defined; this provides sup-
968 port for already existing “character names.”

969 The names selected for the members of the portable character set follow the
970 ISO 8859 {5} and the ISO/IEC 10646-1 {B13} standards. However, several com-
971 monly used UNIX system names occur as synonyms in the list:

- 972 — The historical UNIX system names are used for control characters.
- 973 — The word “slash” is given in addition to “solidus.”
- 974 — The word “backslash” is given in addition to “reverse-solidus.”
- 975 — The word “hyphen” is given in addition to “hyphen-minus.”
- 976 — The word “period” is given in addition to “full-stop.”
- 977 — For digits, the word “digit” is eliminated.
- 978 — For letters, the words “Latin Capital Letter” and “Latin Small Letter” are
979 eliminated.
- 980 — The words “left brace” and “right brace” are given in addition to “left-curly-
981 bracket” and “right-curly-bracket.”
- 982 — The names of the digits are preferred over the numbers to avoid possible
983 confusion between “0” and “O”, and between “1” and “l” (one and the letter
984 ell).

985 The names for the control characters in Table 2-5 were taken from ISO 4873 {3}.

986 The charmap file was introduced to resolve problems with the portability of, espe-
987 cially, `localedef` sources. POSIX.2 assumes that the portable character set is
988 constant across all locales, but it does not prohibit implementations from support-
989 ing two incompatible codings, such as both ASCII and EBCDIC. Such “dual-
990 support” implementations should have all charmaps and `localedef` sources
991 encoded using one portable character set, in effect “cross-compiling” for the other
992 environment. Naturally, charmaps (and `localedef` sources) are only portable
993 without transformation between systems using the same encodings for the port-
994 able character set. They can, however, be transformed between two sets using
995 only a subset of the actual characters (the portable set). However, the particular
996 coded character set used for an application or an implementation does not neces-
997 sarily imply different characteristics or collation: on the contrary, these attri-
998 butes should in many cases be identical, regardless of code set. The charmap pro-
999 vides the capability to define a common locale definition for multiple code sets (the
1000 same `localedef` source can be used for code sets with different extended charac-
1001 ters; the ability in the charmap to define “empty” names allows for characters
1002 missing in certain code sets).

1003 In addition, interest has been expressed in using the charmap concept to provide
1004 the information required for support of multiple character sets. Examples of such
1005 information are encoding mechanisms, string parsing rules, default font informa-
1006 tion, etc. Such extensions are not described here.

1007 The `<escape_char>` declaration was added at the request of the international
1008 community to ease the creation of portable *charmap* files on terminals not imple-
1009 menting the default backslash escape. (This approach was adopted because this
1010 is a new interface invented by POSIX.2. Historical interfaces, such as the shell
1011 command language and `awk`, have not been modified to accommodate this type of
1012 terminal.) The `<comment_char>` declaration was added at the request of the
1013 international community to eliminate the potential confusion between the number
1014 sign and the pound sign.

1015 The octal number notation with no leading zero required was selected to match
1016 those of `awk` and `tr` and is consistent with that used by `localedef`. To avoid
1017 confusion between an octal constant and the backreferences used in `localedef`
1018 source, the octal, hexadecimal, and decimal constants must contain at least two
1019 digits. As single-digit constants are relatively rare, this should not impose any
1020 significant hardship. Each of the constants includes “two or more” digits to
1021 account for systems in which the byte size is larger than 8 b. For example, a
1022 Unicode (ISO/IEC 10646-1 (B13)) system that has defined 16 b bytes may require
1023 six octal, four hexadecimal, and five decimal digits.

1024 The decimal notation is supported because some newer international standards
1025 define character values in decimal, rather than in the old column/row notation.

1026 Multibyte characters are described in the charmap file using “big Endian” nota-
1027 tion for reasons of portability. There is no requirement that the internal
1028 representation in the computer memory be in this same order.

1029 The charmap identifies the coded character sets supported by an implementation.
1030 At least one charmap must be provided, but no implementation is required to pro-
1031 vide more than one. Likewise, implementations can allow users to generate new
1032 charmaps (for instance, for a new version of the 8859 family of coded character

1033 sets), but does not have to do so. If users are allowed to create new charmaps, the
1034 system documentation must describe the rules that apply (for instance: “only
1035 coded character sets that are supersets of ISO/IEC 646 {1} IRV, no multibyte char-
1036 acters, etc.”).

1037 **E.2.5 Locale**

1038 The description of locales is based on work performed in the UniForum Technical
1039 Committee Subcommittee on Internationalization. Wherever appropriate, key-
1040 words were taken from the C Standard [7] or the *X/Open Portability Guide* [B49].

1041 The value used to specify a locale with environment variables is the name
1042 specified as the *name* operand to the `localedef` utility when the locale was
1043 created. This provides a verifiable method to create and invoke a locale.

1044 The “object” definitions need not be portable, as long as “source” definitions are.
1045 Strictly speaking, source definitions are portable only between implementations
1046 using the same character set(s). Such source definitions, if they use symbolic
1047 names only, easily can be ported between systems using different code sets, as
1048 long as the characters in the portable character set (Table 2-4) have common
1049 values between the code sets; this is frequently the case in historical implementa-
1050 tions. Of course, this requires that the symbolic names used for characters out-
1051 side the portable character set be identical between character sets. The definition
1052 of symbolic names for characters is outside the scope of POSIX.2, but is certainly
1053 within the scope of other standards organizations. When such names are stand-
1054 ardized, future versions of POSIX.2 should require the use of these names.

1055 Applications can select the desired locale by invoking the `setlocale()` function (or
1056 equivalent) with the appropriate value. If the function is invoked with an empty
1057 string, the value of the corresponding environment variable is used. If the
1058 environment variable is not set or is set to the empty string, the implementation
1059 sets the appropriate environment as defined in 2.6.

1060 **E.2.5.1 POSIX Locale**

1061 The POSIX Locale is equal to the “C” locale, as specified in POSIX.1 [8]. To avoid
1062 being classified as a C-language function, the name has been changed to the
1063 *POSIX Locale*; the environment variable value can be either “POSIX”, or, for his-
1064 torical reasons, “C”.

1065 The POSIX definitions mirror the historical UNIX system behavior.

1066 The use of symbolic names for characters in the tables does not imply that the
1067 POSIX Locale must be described using symbolic character names, but merely that
1068 it may be advantageous to do so.

1069 Implementations must define a locale as the “default” locale, to be invoked when
1070 no environment variables are set, or set to the empty string. This default locale
1071 can be the POSIX Locale or any other, implementation-defined locale. Some
1072 implementations may provide facilities for local installation administrators to set
1073 the default locale, customizing it for each location. POSIX.2 does not require such
1074 a facility.

1075 **E.2.5.2 Locale Definition**

1076 The decision to separate the file format from the `localedef` utility description
1077 was only partially editorial. Implementations may provide other interfaces than
1078 `localedef`. Requirements on “the utility,” mostly concerning error messages,
1079 are described in this way because they are meant to affect the other interfaces
1080 implementations may provide as well as `localedef`. (This is similar to the phi-
1081 losophy used by POSIX.1 {8} where the descriptions of the `tar` and `cpio` file for-
1082 mats impose requirements on any utilities processing them.)

1083 The text about {POSIX2_LOCALEDEF} does not mean that internationalization is
1084 optional; only that the functionality of the `localedef` utility is. REs, for
1085 instance, must still be able to recognize, for example, character class expressions
1086 such as `[[:alpha:]]`. A possible analogy is with an applications development
1087 environment: while all conforming implementations must be capable of executing
1088 applications, not all need to have the development environment installed. The
1089 assumption is that the capability to modify the behavior of utilities (and applica-
1090 tions) via locale settings must be supported. If the `localedef` utility is not
1091 present, then the only choice is to select an existing (presumably
1092 implementation-documented) locale. An implementation could, for example,
1093 chose to support only the POSIX Locale, which would in effect limit the amount of
1094 changes from historical implementations quite drastically. The `localedef` util-
1095 ity is still required, but would always terminate with an exit code indicating that
1096 no locale could be created. Supported locales must be documented using the syn-
1097 tax defined in 2.5. (This ensures that users can accurately determine what capa-
1098 bilities are provided. If the implementation decides to provide additional capabili-
1099 ties to the ones in 2.5, that is already provided for.)

1100 If the option is present (i.e., locales can be created), then the `localedef` utility
1101 must be capable of creating locales based on the syntax and rules defined in 2.5.
1102 This does not mean that the implementation cannot also provide alternate means
1103 for creating locales.

1104 The octal, decimal, and hexadecimal notations are the same employed by the
1105 `charmap` facility (see 2.4.1). To avoid confusion between an octal constant and a
1106 backreference, the octal, hexadecimal, and decimal constants must contain at
1107 least two digits. As single-digit constants are relatively rare, this should not
1108 impose any significant hardship. Each of the constants includes “two or more”
1109 digits to account for systems in which the byte size is larger than 8 b. For exam-
1110 ple, a Unicode (ISO/IEC 10646-1 {B13}) system that has defined 16 b bytes may
1111 require six octal, four hexadecimal, and five decimal digits. As with the `charmap`
1112 file, multibyte characters are described in the locale definition file using “big
1113 Endian” notation for reasons of portability. There is no requirement that the
1114 internal representation in the computer memory be in this same order.

1115 Since POSIX.2 was developed as an international standard, it adheres wherever
1116 possible to ISO/IEC guidelines for such development. One such guideline is that
1117 characters outside the invariant part of ISO/IEC 646 {1} should not be used in
1118 portable specifications. The backslash character is not in the invariant part; the
1119 number sign is, but with multiple representations: as a number sign and as a
1120 pound sign. As far as general usage of these symbols, they are covered by the
1121 “grandfather clause,” but for newly defined interfaces, the WG15 POSIX working