IEEE P1003.2 Draft 11.2 – September 1991

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All rights reserved as an unpublished work.

This is an unapproved and unpublished IEEE Standards Draft, subject to change.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

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

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

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

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

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

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

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

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

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

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

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

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

{ [( )] }

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

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

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

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

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

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

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

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

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

2.1.2 Grammar Conventions

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

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

— The identifiers for the reserved words of the language are shown with a leading capital letter. (These are terminals in the grammar. Examples: `While`, `Case`.)

— The identifiers for terminals in the grammar are all named with uppercase letters and underscores. Examples: `NEWLINE`, `ASSIGN_OP`, `NAME`.

— The identifiers for nonterminals are all lowercase.

2.1.3 Miscellaneous Conventions

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

— The programs use the syntax and semantics described by the C Standard {7}.

— The programs are merely examples and do not represent the most efficient, or only, means of coding the interface. Implementations may use other programming languages or algorithms, as long as the results are the same as those achieved by the programs in this standard.

— C-language comments are informative and pose no requirements.

Further conventions are presented in:

— Utility Conventions, 2.10, describing utility and application command-line syntax

— File Format Notation, 2.12, describing the notation used to represent utility input and output

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

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

The C language was chosen for many examples because:

— It eliminates any requirement to document a different pseudocode.

— It is a familiar language to many of the potential readers of POSIX. 2.

— It is the language most widely used for historical implementations of the utilities.

2.2 Definitions

2.2.1 Terminology

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

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

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

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

— configure it for conformance with this standard;

— select some of the various optional facilities described by this standard, through customization by local system administrators or operators.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(1) Shell command language

(2) Command arguments

(3) Regular expressions, of various types

(4) Command input language syntax, such as `awk`, `bc`, `ed`, `lex`, `make`, `sed`, and `yacc`. Some of these are so complex that they rival traditional high level languages.

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

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

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

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

2.2.2 General Terms

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

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

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

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

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

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

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

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

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

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

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

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

2.2.2.8 asterisk: The character "*".

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

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

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

2.2.2.11 backquote: The character ‘‘`’’,also known as a *grave accent* .

2.2.2.12 backslash: The character ‘‘\’’,also known as a *reverse solidus* .

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

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

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

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

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

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

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

See also *empty line* (2.2.2.44).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[POSIX .1 {8}]

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

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

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

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

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

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

2.2.2.27 circumflex: The character ‘‘^’’.

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

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

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

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

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

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

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

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

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

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

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

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

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

2.2.2.34 command language interpreter: See 2.2.2.133.

2.2.2.35 directory: A file that contains directory entries.

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

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

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

2.2.2.37 dollar-sign: The character ‘‘$’’.

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

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

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

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

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

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

2.2.2.40 double-quote: The character ‘‘"’’,also known as *quotation-mark* .

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

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

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

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

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

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

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

See also *blank line* (2.2.2.17).

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

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

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

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

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

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

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

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

2.2.2.49 execute: To perform the actions described in 3.9.1.1.

See also *invoke* (2.2.2.79).

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

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

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

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

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

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

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

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

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

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

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

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

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

(1) Specify file permission bits for the file owner class, file group class, and file other class of the file, corresponding to the access permissions, to be returned by *stat* () or *fstat* ().

(2) Be enabled only by explicit user action, on a per-file basis by the file owner or a user with the appropriate privilege.

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

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

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

(1) If a process has the appropriate privilege:

(a) If read, write, or directory search permission is requested, access is granted.

(b) If execute permission is requested, access is granted if execute permission is granted to at least one user by the file permission bits or by an alternate access control mechanism; otherwise, access is denied.

(2) Otherwise:

(a) The file permission bits of a file contain read, write, and execute/search permissions for the file owner class, file group class, and file other class.

(b) Access is granted if an alternate access control mechanism is not enabled and the requested access permission bit is set for the class (file owner class, file group class, or file other class) to which the process belongs, or if an alternate access control mechanism is enabled and it allows the requested access; otherwise, access is denied.

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

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

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

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

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

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

513 **2.2.2.59 file mode:** An object containing the file permission bits and other
514 characteristics of a file, as described in POSIX.1 {8} 5.6.1. [POSIX.1 {8}]

515 **2.2.2.60 file mode bits:** A file's file permission bits, set-user-ID-on-execution bit
516 (S_ISUID), and set-group-ID-on-execution bit (S_ISGID) (see POSIX.1 {8} 5.6.1.2).

517 **2.2.2.61 filename:** A name consisting of 1 to {NAME_MAX} bytes used to name a
518 file.

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

523 **2.2.2.62 filename portability:** A concept of the underlying system, as follows.
524 [POSIX.1 {8}]

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

527 **2.2.2.63 file offset:** The byte position in the file where the next I/O operation
528 begins.

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

533 **2.2.2.64 file other class:** The property of a file indicating access permissions for
534 a process related to the process's user and group identification.

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

537 **2.2.2.65 file owner class:** The property of a file indicating access permissions
538 for a process related to the process's user identification.

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

541 **2.2.2.66 file permission bits:** Information about a file that is used, along with
542 other information, to determine if a process has read, write, or execute/search per-
543 mission to a file.

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

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

549 **2.2.2.67 file serial number:** A per-file-system unique identifier for a file.

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

551 **2.2.2.68 file system:** A collection of files and certain of their attributes.

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

554 **2.2.2.69 file times update:** A concept of the underlying system, as follows.
555 [POSIX .1 {8}]

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

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

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

575 **2.2.2.70 file type:** See *file* in 2.2.2.54.

576 **2.2.2.71 filter:** A command whose operation consists of reading data from stan-
577 dard input or a list of input files and writing data to standard output.

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

579 **2.2.2.72 foreground process:** A process that is a member of a foreground pro-
580 cess group. [POSIX .1 {8}]

581 **2.2.2.73 foreground process group:** A process group whose member processes
582 have certain privileges, denied to processes in background process groups, when
583 accessing their controlling terminal.

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

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

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

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

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

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

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

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

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

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

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

See also *execute* (2.2.2.49).

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

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

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

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

620 **2.2.2.82 link:** See *directory entry* in 2.2.2.36.

621 **2.2.2.83 link count:** The number of directory entries that refer to a particular
622 file. [POSIX .1 {8}]

623 **2.2.2.84 locale:** The definition of the subset of a user's environment that
624 depends on language and cultural conventions; see 2.5.

625 **2.2.2.85 login:** The unspecified activity by which a user gains access to the
626 system.

627 Each login shall be associated with exactly one login name. [POSIX .1 {8}]

628 **2.2.2.86 login name:** A user name that is associated with a login. [POSIX .1 {8}]

629 **2.2.2.87 mode:** A collection of attributes that specifies a file's type and its access
630 permissions.

631 See *file access permissions* in 2.2.2.55. [POSIX .1 {8}]

632 **2.2.2.88 multicharacter collating element:** A sequence of two or more charac-
633 ters that collate as an entity.

634 For example, in some coded character sets, an accented character is represented
635 by a (nonspacing) accent, followed by the letter. Another example is the Spanish
636 elements ''ch''and ''ll.''

637 **2.2.2.89 negative response:** An input string that matches one of the responses
638 acceptable to the LC_MESSAGES category keyword `noexpr`, matching an
639 extended regular expression in the current locale.

640 See 2.5.

641 **2.2.2.90 <newline>:** A character that in the output stream shall indicate that 1
642 printing should start at the beginning of the next line.

643 The `<newline>` shall be the character designated by `'\n'` in the C language
644 binding. It is unspecified whether this character is the exact sequence transmit-
645 ted to an output device by the system to accomplish the movement to the next
646 line.

647 **2.2.2.91 NUL :** A character with all bits set to zero.

648 **2.2.2.92 null string:** See *empty string* in 2.2.2.45.

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

2.2.2.93 number-sign: The character ‘‘#’’.

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

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

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

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

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

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

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

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

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

2.2.2.99 parent directory:

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

(2) When discussing other types of files, a directory containing a directory entry for the file under discussion.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.2.2.107 period: The character ‘‘**.**’’.

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

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

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

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

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

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

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

For a filename to be portable across conforming implementations of this standard, it shall consist only of the following characters:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 . _ -
```

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

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

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