IEEE Standards Interpretations for IEEE Std 1003.2-1992

Circuits and Devices

Communications Technology

IEEE Computer Society

Sponsored by the
Portable Applications Standards Committee

Electromagnetics and Radiation

Energy and Power

Industrial Applications

Signals and Applications

Standards Coordinating Committees

IEEE Std 1003.2-1992/INT, March 1994 Edition

IEEE

Published by the Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017, USA.

May 2, 1994

SH17178

IEEE Std 1003.2-1992/INT
March 1994 Edition

IEEE Standards Interpretations for IEEE Std 1003.2-1992

Sponsor

Portable Applications Standards Committee
of the
IEEE Computer Society

Abstract: The Portable Applications Standards Committee of the IEEE Computer Society carried out a series of analyses of various problems encountered by users of IEEE Std 1003.2-1992, IEEE Standard for Information Technology—Portable Operating System Interface (POSIX)—Part 2: Shell and Utilities. The results of its deliberations are presented in this document. The intent is to give the POSIX community reasonable ways of interpreting unclear portions of the standard.

Keywords: computer, computer environments, interfaces, interpretations, portable operating system interface, POSIX, shell, utilities

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

ISBN 1-55937-411-X

IEEE Standards Interpretations

Occasionally, questions may arise regarding the meaning of portions of standards as they relate to specific applications. Such requests for interpretations should ask for clarifications of the exact nature of the contents of the standard.

Questions relating to such interpretations are reviewed and evaluated by a balance of members representing the specific committee interests. This officially formed interpretations subgroup creates and circulates a draft interpretation among its members. It transmits a final interpretation to the party initiating the request only after consensus has been achieved. The interpretation is also given to the standards-developing committee to consider addressing it in a supplement or the next revision to the standard.

Interpretations are issued to explain and clarify the intent of the standard and are not intended to constitute an alteration to the original standard or to supply consulting information. The interpretations subgroup cannot make new rules to fit situations not yet covered in the standard, even if the investigations of the subgroup lead it to conclude that the requirement is incomplete or in error. Changes to the standard are made only through revisions or supplements to the standard.

It is recognized that requests are frequently received that are partially or totally requests for information rather than requests for an interpretation. It is inappropriate to issue an official interpretation to answer such requests. The interpretations subgroup may, however, find from its research that the literal printing of the standards text is not identical to that approved by the standards developers and may issue an editorial correction as a part of its interpretation.

The original interpretations requests in this document have been lightly edited to remove extraneous matter and to focus on the problem presented. Some illustrations have been redrawn for publication. With these exceptions, requests are in the form received.

Procedure for Requesting an Interpretation

Requests for interpretations should be addressed to

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

Requests for interpretations should include

a) The specific designation of the standard, including the year of publication.
b) The specific subsection being questioned.
c) The applicable conditions for the case in question.

Line drawings should be black ink or excellent black pencil originals. Photos should be black-and-white glossy prints. These illustrations must be reproduced for committee circulation and eventually will be used to supplement the text of the next edition. Clear diagrams and pictures will make the work of interpretation easier and more valuable to users.

Requests, including all supplementary material, must be in a form that is easily reproducible. If suitable for consideration, requests will be sent to the interpretations subgroup. After consideration by the subcommittee, which may involve many exchanges of correspondence, the inquirer will be notified of the decision of the subgroup. Decisions will be published from time to time in cumulative form and may be ordered from the IEEE.

Introduction

After approval and publication of a standard, questions may arise regarding the meaning of portions of standards as they relate to users of the standard. In the case of IEEE Std 1003.2-1992, these questions arose from both system implementors and application authors. When a need for interpretation is brought to the attention of the IEEE, the Institute initiates action to prepare appropriate responses.

IEEE Std 1003.2-1992 (POSIX.2) was developed by the Portable Applications Standards Committee (PASC) of the IEEE Computer Society. Each interpretation request is forwarded to the members of the PASC interpretations group and a subgroup is invited to generate a draft interpretation for review. The draft interpretation is made available to the whole interpretations group for review and comment. When consensus has been reached within the interpretations group, the IEEE transmits the final interpretation to the party initiating the request.

It is recognized that requests are frequently received that are partially or totally requests for information rather than requests for interpretation. The IEEE considers it inappropriate to issue an official interpretation to answer such requests. This includes requests for advice as to the means by which particular facilities described in the standard could be implemented.

Every IEEE standard is subjected to review at least every five years for revision or reaffirmation. The revision process takes input from a wide variety of sources in order to develop a revised standard. One of these sources is the interpretations that have been issued against the extant version of the standard. It is not to be expected that the text of the interpretation will appear in the revised standard, but rather that the revision will address the issues raised by the interpretation request through the consensus process. The revision process is allowed to resolve ambiguities in the original standard through textual change; this capability is not allowed within the interpretations process.

This document contains a compilation of all of the interpretations completed against IEEE Std 1003.2-1992 as of February 1994. The IEEE will provide details of interpretation requests received that have not yet appeared in a publication. Where a formal interpretation has been completed, this will include the text of the interpretation.

At the time that this compilation of interpretations was published, the following members had participated in the IEEE Std 1003.2-1992 interpretations group:

Portable Applications Standards Committee

Chair: Jim Isaak
Vice-Chair, Interpretations: Andrew Josey

Keith Bostic
Mark Brown
Dawn Burnett
Don Cragun
Dave Decot
John Farley
Mark Funkenhauser
Jeff Haemer
Eric Horner
Hal Jespersen
William King
Bob Korn
David Korn
Shane McCarron
Gary Miller
Martha Nalebuff
David Rowley
Thomas Shem
Scott Sutter
Marc Teitelbaum
Alex White
David Willcox
David Williams
Jeff Zado

Mary Lynne Nielsen
IEEE Standards Project Editor

Interpretation Number:	**1**
Topic:	Symbolic limit macros
Relevant Clauses:	2.13
Classification:	Editorial defect

Interpretation request

In B.2, the standard states:

> The following subclauses list the names of macros that C-language applications can use to obtain minimum and current values for limits defined in 2.13.1.

Then in B.2.3, Table B-5 lists C-language macros with names strikingly similar to POSIX.2 optional facility configuration parameters, defined in 2.13.2. Table B-5 misses the configuration parameter {POSIX2_C_BIND}.

Is the intent of the standard that all the variables in 2.13.2 as well as 2.13.1 should have corresponding variables in the C binding, since seven of these variables are listed in Table B-5?

Interpretation for IEEE Std 1003.2-1992

As the standard does not list _POSIX2_C_BIND among the symbols in Table B-5, a conforming implementation is not required to support this symbol in the manner described by B.2.3. This should not be construed to prohibit conforming implementations from supporting _POSIX2_C_BIND in the manner described, as an extension. Concerns about this are being referred to the sponsor.

Rationale for interpretation

In B.2, the intent of the standard is made clear, that the symbols from 2.13.1 are to be listed here as well. It appears that the exclusion of this one symbol is an editing defect in the standard.

Interpretation Number: **2**
Topic: Regular expressions (REs)
Relevant Clauses: B.5.2
Classification: No change required

Interpretation request

In B.5.2, the standard states that the *re_nsub* member of the *regex_t* structure represents the number of parenthesized subexpressions found in *pattern.*

The standard then states (page 728, lines 328–336) that

> the *pmatch* argument shall point to an array with at least *nmatch* elements, and *regexec()* shall fill in the elements of that array with offsets of the substrings of *string* that correspond to the parenthesized subexpressions of *pattern*: The byte offset of the beginning shall be `pmatch[i].rm_so`, and `pmatch[i].rm_eo` shall be one greater than the byte offset of the end of substring *i.* (Subexpression *i* begins at the *i*th matched open parenthesis, counting from 1.) Offsets in *pmatch*[0] shall identify the substring that corresponds to the entire regular expression.

Thus, if *pmatch*[] contains *nmatch* elements, it can only hold *nmatch–1* parenthesized subexpressions of *string*, since *pmatch*[0] represents the entire regular expression.

> The standard also states that "if there are more than *nmatch* subexpressions in *pattern* (*pattern* itself counts as a subexpression), then *regexec()* [...] shall record only the first *nmatch* substrings."

Lines 336–338 appear to contradict lines 328–336; the latter talks about parenthesized subexpressions, while the former mentions plain subexpressions. Is the intent of the standard to allow the *re_nsub* member to include the subexpression representing the entire regular expression in the count (since it is considered a subexpression on page 728, lines 328–336), or does it only count explicitly parenthesized subexpressions? This may be the easiest way to rectify the ambiguity.

Interpretation for IEEE Std 1003.2-1992

The subexpression representing the entire RE is to be included in the count represented in the *re_nsub* member. No change in wording is necessary.

Rationale for interpretation

The section quoted in the request contains the phrase "(*pattern* itself counts as an expression)," which the committee considers key to interpreting this apparent conflict.

Interpretation Number:	**3**
Topic:	dd
Relevant Clauses:	4.16.2
Classification:	Ambiguous

Interpretation request

In 4.16 (page 245, lines 3053–3054), the standard states that "if the swab conversion is specified, each pair of input data bytes shall be swapped. If there are an odd number of bytes in the input block, the results are unspecified."

Then, on page 247, lines 3133–3135, the standard contradicts itself, saying the swab conversion means "swap every pair of input bytes. If the current input record is an odd number of bytes, the last byte in the input record shall be ignored."

Which of these two passages is valid?

Interpretation for IEEE Std 1003.2-1992

This is a conflict in the standard (and neither section matches historical practice). In this case, the implementation is allowed to choose either behavior. Concerns about this have been referred to the sponsor of the standard for possible amendment.

Rationale for interpretation

None.

Interpretation Number: **4**
Topic: dd
Relevant Clauses: 4.16.4
Classification: Ambiguous

Interpretation request

In 4.16.4, the standard states that the `conv=swab` conversion should "swap every pair of bytes. If the current input record is an odd number of bytes, the last byte in the input record shall be ignored."

Does the standard mean that the last byte is ignored with respect to the conversion, or ignored completely (thus making the output file shorter than the original)?

Interpretation for IEEE Std 1003.2-1992

The resolution for this request is within the same scope as Interpretation #3, and that resolution shall apply.

Rationale for interpretation

None.

Interpretation Number:	**5**
Topic:	dd
Relevant Clauses:	4.16.2
Classification:	No change required

Interpretation request

In 4.16.2, the standard states that:

> (2) If the input block is shorter than the specified input block size and the `sync` conversion is specified, null bytes shall be appended to the input data up to the specified size. (If either `block` or `unblock` is also specified, `<space>` characters shall be appended instead of null bytes.) The remaining conversions and output shall include the pad characters as if they had been read from the input.
>
> [...]
>
> (6) The data resulting from input or conversion or both shall be aggregated into output blocks of the specified size. After the end of input is reached, any remaining output shall be written as a block without padding if `conv=sync` is not specified; thus, the final output block may be shorter than the output block size.

In 4.16.4, the various `conv=` conversions are specified:

> `block` Treat the input as a sequence of `<newline>`-terminated or end-of-file-terminated variable-length records independent of the input block boundaries. Each record shall be converted to a record with a fixed length specified by the conversion block size. Any `<newline>` shall be removed from the input line; `<space>`s shall be appended to lines that are shorter than their conversion block size to fill the block. Lines that are longer than the conversion block size shall be truncated to the largest number of characters that will fit into that size; the number of truncated lines shall be reported (see 4.16.6.2).
>
> [...]
>
> `sync` Pad every input block to the size of the `ibs=` buffer, appending null bytes. If either `block` or `unblock` is also specified, append `<space>` characters, rather than null bytes.

The standard does not specify what happens to the output if the `conv=sync` conversion is specified, most notably in the case when `conv=block` is also specified. Does the output block get padded with `<space>` characters in this case, or is no output padding done? The definition of the `conv=sync` conversion does not describe output buffering; the only reference to it is in 4.16.2, cited above.

Interpretation for IEEE Std 1003.2-1992

The descriptions of `conv=sync` and `conv=block` are clear and orthogonal, `conv=sync` referring to input only and `conv=block` to output only. Hence, specifying `conv=sync` has no effect on how output is handled, other than its specific role in modifying input.

Rationale for interpretation

None.

Interpretation Number:	**6**
Topic:	`dd`
Relevant Clauses:	4.16.5.4, 2.11.5.4
Classification:	Editorial defect

Interpretation request

In 4.16.5.4, the standard states that "for SIGINT, the `dd` utility shall write status information to standard error before exiting. It shall take the standard action for all other signals; see 2.11.5.4."

In this context, what does "exit" mean? Does this literally mean the use of the *exit()* function or does it really mean "process termination"? If the *exit()* function is mandated, then `dd` contradicts the spirit of 2.11.5.4 and the behavior of all other POSIX.2 utilities: the parent process cannot tell that `dd` terminated due to a signal.

Interpretation for IEEE Std 1003.2-1992

The standard states that the "utility shall write status information to standard error before exiting" when the utility receives a SIGINT. While this can possibly be construed to mean "SIGINT causes the utility to write status information before it exits, whenever it does exit," the proper reading of the sentence is "upon receipt of SIGINT, the utility will exit, writing status information before doing so."

Additionally, it is possible to read this phrase as requiring the utility to use *exit()*. The use of the words "before exiting" in this subclause of the standard does not require the implementation to use *exit()*.

Concerns about the wording of this part of the standard have been forwarded to the sponsor.

Rationale for interpretation

None.

Interpretation Number: 7
Topic: ed
Relevant Clauses: 4.20.5.4
Classification: No change required

Interpretation request

In 4.20.5.4, the standard states:

> The ed utility shall take the standard action for all signals (see 2.11.5.4), with the following exceptions:
>
> SIGINT [...]
>
> SIGHUP [...]

The SIGQUIT signal is not explicitly mentioned in the list that follows the above statement. Does the standard really mean that ed cannot clean up or otherwise handle a SIGQUIT signal?

According to 1.1:

> The facilities to be provided are based on the historical models of the following documents: the *System V Interface Definition* {B36}: the BSD User Manual {B45}; the *X/Open Portability Guide* {B48}; and documentation for the KornShell {B37}. Emphasis is 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.

Disallowing the ed utility from intercepting SIGQUIT is not historical practice, and is likely to cause problems for existing users of existing utilities. Is the omission of SIGQUIT from the list in 4.20.5.4 simply an oversight? Any production-level interactive command should ignore a SIGQUIT signal.

Interpretation for IEEE Std 1003.2-1992

The standard clearly requires the utility to perform as if it had taken the standard action upon receipt of SIGINT, by reference to 2.11.5.4, option 3. Thus, the utility is allowed to perform cleanup actions as long as it meets the requirements of 2.11.5.4.

Rationale for interpretation

None.

Interpretation Number:	**8**
Topic:	`sort`
Relevant Clauses:	2.9.1.4
Classification:	No change required

Interpretation request

In 2.9.1.4, the standard states that:

> When an attempt is made to create a file that already exists, the action shall depend on the file type:
>
> (1) For directories and FIFO special files, the attempt shall fail and the utility shall either continue with its operation or exit immediately with a nonzero status, depending on the description of the utility.

Does the standard really mean what it says? Will every utility that creates an output file need to first check that there is not a file of type FIFO with that name before it *open()*s it for writing?

If so, `mkfifo foo; sort -o foo` file will fail. Being able to use a FIFO as a pipe with a name is useful behavior. It allows applications to connect utilities together in more complex ways than simple pipelines.

Interpretation for IEEE Std 1003.2-1992

The quoted text from 2.9.1.4 is modified by 2.9.1, lines 3360–3361, "utility and function description statements override these defaults when appropriate."

The `sort -o` option is described in 4.58.6.3 as "If the `-o` option is in effect, the sorted output shall be placed in the file . . ." This overrides 2.9.1.4, requiring that the output be placed in the file, as it does not speak of requiring creation of said file.

Rationale for interpretation

None.

Interpretation Number: **9**
Topic: LC_CTYPE
Relevant Clauses: E.3.5.3
Classification: No change required

Interpretation request

In 3.5.3, the standard states that:

> This variable [LC_CTYPE] shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters), which characters are defined as letters (character class `alpha`) and `<blank>`s (character class `blank`), and the behavior of character classes within pattern matching. Changing the value of LC_CTYPE after the shell has started shall not affect the lexical processing of shell commands in the current shell execution environment or its subshells (see 3.12).

The standard also states that the **LANG** variable "shall provide a default value for the **LC_*** variables, as described in 2.6" and that the **LC_ALL** variable "shall interact with the **LANG** and **LC_*** variables as described in 2.6."

In 2.6, the standard summarizes the meanings of these variables:

LANG	This variable shall determine the locale category for any category not specifically selected via a variable starting with **LC_**. **LANG** and the **LC_** variables can be used by applications to determine the language for messages and instructions, collating sequences, date formats, etc. Additional semantics of this variable, if any, are implementation defined.
LC_ALL	This variable shall override the value of the **LANG** variable and the value of any of the other variables starting with **LC_**.
[...]	
LC_CTYPE	This variable shall determine the locale category for character handling functions. This environment variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters), the classification of characters (e.g., alpha, digit, graph), and the behavior of character classes. Additional semantics of this variable, if any, are implementation defined.

Does changing **LC_ALL** (or **LANG** if **LC_CTYPE** is not set) affect the lexical processing of shell commands in the current shell execution environment? Is it the intent of the standard that any changes to environment variables that cause a new **LC_TYPE** to be used shall be ignored by the shell once it has started execution?

An implementation of `sh` must use the locale specified in **LC_CTYPE** when reading a script. For example, `isalpha/isalnum` is used to parse variable names.

Consider this simple command:

`FO<`*O-umlaut*`>=BAR cmd`

If `isalnum('<`*O-umlaut*`>')`, then this will parse as a variable assignment; otherwise, it is argument 0. Similarly, `cmd` will be subject to alias expansion in the former case. There is no need to validate variable names at other times. In such an implementation, changing **LC_CTYPE** causes no problems.

What are the problems with the following commands:

```
LANG=locale-with-O-umlaut
FO<O-umlaut>=BAR
```

Then consider this sequence of commands:

```
[ -n "$FO<O-umlaut> " ] && alias echo=:
echo foo
```

In both cases, the parsing of the second line is determined by the execution of the first line. Traditional implementations execute the first line, then parse and execute the second line. What would a compiler do?

On the other hand, if they where embedded in {...} or any other shell compound command, they would both be parsed before being executed. So there are two cases where behavior is poorly defined or context dependent.

The behavior of setting the **LC_CTYPE** should be made undefined. Changing **LANG** in an interactive shell is a reasonable action, and an implementation may immediately change all locales with no problems. Having all but one locale change, and just in the shell, is unintuitive and not required.

Interpretation for IEEE Std 1003.2-1992

The standard clearly states that changes to **LC_CTYPE** shall not take effect within the current shell execution environment. This is discussed in the rationale, E.3.5.3.

Rationale for interpretation

None.

Interpretation Number: **10**
Topic: mailx
Relevant Clauses: 4.40.5.4
Classification: No change required

Interpretation request

In 4.40.5.4, the standard states that, except for the SIGINT signal, "the `mailx` utility shall take the standard action for all other signals; see 2.11.5.4'."

Does the standard really mean that `mailx` cannot clean up or otherwise handle a SIGQUIT signal?

According to 1.1:

> The facilities to be provided are based on the historical models of the following documents: the *System V Interface Definition* {B36}; the BSD User Manual {B45}; the *X/Open Portability Guide* {B48}; and documentation for the KornShell {B37}. Emphasis is 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.

Any production-level interactive command should ignore a SIGQUIT signal. Is the omission of SIGQUIT from the list in 4.40.5.4 simply an oversight?

Interpretation for IEEE Std 1003.2-1992

As stated in 2.11.5.4, option 3 permits the result of the execution of the utility to be as if default actions had been taken. This should not prevent the implementation from performing cleanup operations.

Rationale for interpretation

None.

Interpretation Number: **11**
Topic: `mailx`
Relevant Clauses: 4.40.7
Classification: Unaddressed issue

Interpretation request

In 4.40.7, the standard states the order in which `mailx` initialization occurs:

> At startup time, `mailx` shall take the following steps in sequence:
>
> (1) Establish all variables at their stated default values.
>
> (2) Process command-line options, overriding corresponding default values.
>
> (3) Import any of the **DEAD**, **EDITOR**, **MBOX**, **LISTER**, **PAGER**, **SHELL**, or **VISUAL** variables that are present in the environment, overriding the corresponding default values.
>
> (4) Read `mailx` commands from an unspecified system startup file, unless the `-n` option is given, to initialize any internal `mailx` variables and aliases.
>
> (5) Process the startup file of `mailx` commands named in the user **MAILRC** variable.

The standard does not indicate when mailboxes are read in this sequence. It is reasonable to assume that the standard intends to describe historical `mailx` operations unless it explicitly distinguishes itself. Can the quiescence of the standard on the matter be interpreted as implying the historical behavior? The behavior is as above, with the following steps added:

> (4a) Read the system mailbox if in *read* mode and if the `-f` option is not specified.
>
> (6) Read the mailbox specified with the `-f` option.

Interpretation for IEEE Std 1003.2-1992

The standard does not address this issue (allowing an implementation to process unspecified actions in any order). Concern over the specification in this area has been forwarded to the sponsor of the standard.

Rationale for interpretation

None.

Interpretation Number:	**12**
Topic:	`patch`
Relevant Clauses:	5.22.7.1
Classification:	No change required

Interpretation request

In 5.22.7, the standard states that "each patch shall contain zero or more lines of filename identification in the format produced by `diff -c` (see 4.17), and one or more sets of `diff` output, which customarily are called 'hunks'."

Since there is no restriction placed on the sets of `diff` output, is the following a valid patch?

```
*** dummy~ Thu Nov 19 14:55:01 1992
--- dummy Thu Nov 19 14:55:01 1992
18d17
< Ababa
20c19,21
< abacus
---
> Ababa
> abcaus
> test
```

If not, and "hunks" are only intended to be context `diff` hunks, then this subclause is incomplete since it fails to specify the format of regular `diff` and `ed` patches.

Interpretation for IEEE Std 1003.2-1992

The utility is required to interpret the information within each hunk as described in 5.22.7.3, lines 3591–3594.

Rationale for interpretation

None.

Interpretation Number:	**13**
Topic:	`patch`
Relevant Clauses:	5.22.7.1
Classification:	Unaddressed issue

Interpretation request

In 5.22.7.1, the standard states that the patch utility shall recognize the following expressions:

`***` *filename timestamp*	The patches arose from *filename*.
`---` *filename timestamp*	The patches should be applied to *filename*.

In 5.22.7.2, the standard then describes what occurs when `patch` finds these lines (in point #1):

> then [`patch` shall] test for the existence of both files in the current directory (or the directory specified with the `-d` option). If both files exist, the `patch` utility shall assume that no pathname can be obtained from this step.

If only one of these files exists, which one is used to obtain the pathname? One would expect the existing file; however, the standard does not explicitly state it.

Interpretation for IEEE Std 1003.2-1992

The standard does not address this issue, and a conforming implementation may take any behavior in this area. Concern over the specification in this area has been forwarded to the sponsor of the standard.

Rationale for interpretation

None.

Interpretation Number:	**14**
Topic:	`patch`
Relevant Clauses:	5.22.7.1
Classification:	Editorial defect

Interpretation request

In 5.22.5.3, the standard states that the **LC_TIME** variable "shall determine the locale for recognizing the format of file time stamps written by the `diff` utility in a context-diff input file.'

In 5.22.7.1, the standard states that the `patch` utility shall recognize the following expressions:

`***` *filename timestamp*	The patches arose from *filename*.
`---` *filename timestamp*	The patches should be applied to *filename*.

What happens when `patch` detects lines that look like *** *filename* and --- *filename*, but the timestamp fields are filled with gibberish (because, for example, they were produced in a different locale)? Are the hunks that follow these lines ignored, since the sanity check by `patch` on these filename identification lines failed?

Interpretation for IEEE Std 1003.2-1992

The implementation is not required to do any processing of the *timestamp* field of the filename identification line(s). The implementation is required to process a hunk according to the standard, regardless of the state or contents of the *timestamp* field.

Rationale for interpretation

None.

Interpretation Number:	**1 5**
Topic:	`patch`
Relevant Clauses:	5.22.7.3
Classification:	Unaddressed issue

Interpretation request

In 5.22.7.3, the standard states:

> For each hunk, the `patch` utility shall begin to search for the place to apply the patch at the line number at the beginning of the hunk, plus or minus any offset used in applying the previous hunk. If lines matching the hunk context are not found, the `patch` utility shall scan both forwards and backwards at least 1000 B for a set of lines that match the hunk context.
>
> If no such place is found, and it is a context diff, then another scan shall take place, ignoring the first and last lines of context. If that fails, the first two and last two lines of context shall be ignored and another scan shall be made.

However, the standard neglects to discuss when an attempt at reversing the patch is performed (even though support for reverse attempts is intended, since the description of the -R option in 5.22.3 mentions this behavior) and never states exactly what should be done with the patch once the right place is found.

When should a reverse patch be attempted? Is it performed after three scans are completed, or after each scan?

Interpretation for IEEE Std 1003.2-1992

The standard is silent on how a request (via -R) to reverse a patch shall be processed; the standard thus allows any behavior by a conforming implementation. Concern over the wording of this area has been forwarded to the sponsor of this standard.

Rationale for interpretation

None.

Interpretation Number:	**16**
Topic:	pax
Relevant Clauses:	4.48.2
Classification:	No change required

Interpretation request

In 4.48.2, the standard states that, when using pax in *read* mode, "any of the various names in the archive that represent a file can be used to select the file for extraction."

Does this mean that if the following steps were performed:

```
echo testing >a
ln a b
pax -wf foo -x ustar a b
```

and then, in a clean directory, the following command were performed:

```
pax -rf foo b
```

the file b will be created, with the contents of the original file a? Historical versions of tar will not extract the file, but will complain about not being able to link to the file a. Given the extended tar format defined by POSIX.1, it would be prohibitively expensive to re-read the archive to locate the contents of file a. This is not a problem with cpio format, since the contents of the file a may appear twice.

According to 1.1:

> The facilities to be provided are based on the historical models of the following documents: the *System V Interface Definition* {B36}; the BSD User Manual {B45}; the *X/Open Portability Guide* {B48}; and documentation for the KornShell {B37}. Emphasis is 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.

Does the standard mean to document historical practice, or is this an example of item (d) in the above list?

Interpretation for IEEE Std 1003.2-1992

The standard implies that the implementation must rewind the archive and restore the contents from the original file. The relevant text is from 4.48.2, lines 7705–7708:

> If the selected archive format supports the specification of linked files, it shall be an error if these files cannot be linked when the archive is extracted. Any of the various names in the archive that represent a file can be used to select the file for extraction.

The first sentence pertains to situations in which the file is extracted but the link cannot be created, because the second sentence supports selection of any link name for use in extracting the actual file.

Rationale for interpretation

Historical practice does not apply in this case because the standard does not attempt to specify the behavior of historical `tar` utilities. A new utility, `pax`, is being specified.

Interpretation Number:	**17**
Topic:	`pax`
Relevant Clauses:	4.48.3
Classification:	No change required

Interpretation request

In 4.48.3, the standard states that the `-k` option to `pax` means "prevent the overwriting of existing files" and the `-n` option means:

> Select the first archive member that matches each *pattern* operand. No more than one archive member shall be matched for each pattern (although members of type directory shall still match the file hierarchy rooted at that file).

If both the `-k` and `-n` options are specified, and the first file matched would cause an existing file to be overwritten, has the offending file been selected and, thus, the offending *pattern* no longer eligible for matching?

Interpretation for IEEE Std 1003.2-1992

The wording of the `-n` option very simply states "select the first archive member that matches the pattern." Therefore, even though the file was not written due to the presence of the `-k` option, the file has been selected and the pattern should not be reused.

Rationale for interpretation

None.

Interpretation Number: **18**
Topic: pax
Relevant Clauses: 4.48.2
Classification: Editorial defect

Interpretation request

In 4.48.2, the standard states that "if an extracted file is of type directory, the file hierarchy rooted at that file shall be extracted as well."

And then:

> . . . if intermediate directories are necessary to extract an archive member, `pax` shall perform actions equivalent to the POSIX.1 {8} *mkdir()* function, called with the following arguments:
>
> — The intermediate directory used as the *path* argument.
>
> — The value of the bitwise inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO as the *mode* argument.

Assuming there is a `ustar` archive with the members `foo/bar` and `foo/blat`, and they are extracted using the command `pax -r foo <paxfile`. Now, if `foo/bar` and `foo/blat` are extracted explicitly, everything is fine because directory `foo` is created with execute permissions.

However, if only `foo` is extracted (and thus its descendants as well), there are problems because without the `-p e` or `-p p` flags set, the execute permissions will not be set on directory `foo`: "[...] otherwise, the attribute shall be determined as part of the normal file creation action (see 2.9.1.4)." In 2.9.1.4, the standard states that:

> the file's permission bits are set to:
>
> S_IROTH|S_IWOTH|S_IRGRP|S_IWGRP|S_IRUSR|S_IWUSR
>
> (see Section 5.6.1.2 of POSIX.1 {8}) except that the bits specified by the file mode creation mask of the process are cleared.

In this subclause, directories are not treated differently than regular files. The `mkdir` utility explicitly adds the needed execute permissions, as does `pax` on intermediate directories.

Is the "spirit" of the standard to create explicitly named directories in this manner, or may they have their execute permissions set?

Interpretation for IEEE Std 1003.2-1992

The standard does not require that the directory have the mode specified until after `pax` has exited. This allows the implementation to leave a more useful mode in place until it completes it processing, then change it to the specified mode. See E.4.48.

Concern over the wording of this clause of the standard has been forwarded to the sponsor of the standard.

Rationale for interpretation

None.

Interpretation Number:	**19**
Topic:	`pax`
Relevant Clauses:	4.48.3
Classification:	Unaddressed issue

Interpretation request

In 4.48.3, the standard describes the `-i` option:

> Interactively rename files or archive members. For each archive member matching a *pattern* operand or file matching a file operand, a prompt shall be written to the file `/dev/tty`. The prompt shall contain the name of the file or archive member, but is otherwise unspecified. A line shall then be read from `/dev/tty`. If this line is blank, the file or archive member shall be skipped. If this line consists of a single period, the file or archive member shall be processed with no modification to its name. Otherwise, its name shall be replaced with the contents of the line. The `pax` utility shall immediately exit with a nonzero exit status if end-of-file is encountered when reading a response or if `/dev/tty` cannot be opened for reading and writing.

The standard previously stated, in 4.48.2, that "if the selected archive format supports the specification of linked files, it shall be an error if these files cannot be linked when the archive is extracted. Any of the various names in the archive that represent a file can be used to select the file for extraction."

If an attempt is made to extract a file from a `ustar` archive based on a linkname to a previous member of the archive that had been renamed using the `-i` option, is the file linked to the new name, or is this an error condition?

Interpretation for IEEE Std 1003.2-1992

The standard is silent on this issue, allowing any behavior in this area. Concern over the wording of this subclause has been forwarded to the sponsor of the standard.

Rationale for interpretation

None.

Interpretation Number:	**21**
Topic:	`sh`
Relevant Clauses:	3.13.1
Classification:	Editorial defect

Interpretation request

In 3.13.1, lines 1390–1392, the standard states that "special characters can be escaped to remove their special meaning by preceding them with a `<backslash>`." The standard defines these "special characters" to be "?," "*," and "[."

Thus, it appears that special characters within [] are not within the scope of the statement on lines 1390–1392. Thus, the statement in 2.8.3.2, lines 2873–2876, "the special characters . * [\ (period, asterisk, left-bracket, and backslash, respectively) shall lose their special meaning within a bracket expression," presumably holds.

This implies that the pattern bracket expression `[a\-z]` should match the letter `a`, and `<backslash>` through `z`, as it does in regular expressions. This disagrees with historical practice, where this pattern matches `a`, `-`, and `z`. Is the intent to disable historical practice, or can lines 1390–1392 be interpreted to read ". . . special characters, including special characters within a pattern bracket expression"?

Interpretation for IEEE Std 1003.2-1992

The standard is specific in its requirement that RE behavior apply to bracket expressions with the one noted exception. The example in the request does not fulfill this exception; hence, it must be interpreted according to the RE rules (2.8.3.2). Concern over the wording of this subclause has been forwarded to the sponsor of the standard.

Rationale for interpretation

None.

Interpretation Number:	**22**
Topic:	talk
Relevant Clauses:	5.28.2
Classification:	No change required

Interpretation request

In 5.28.2, the standard states that "typing other nonprintable characters shall cause implementation-defined sequences of printable characters to be written to the terminal of the recipient" and that "typing characters from LC_CTYPE classifications `print` or `space` shall cause those characters to be sent to the terminal of the recipient."

If {POSIX2_LOCALEDEF} is defined, a malicious user could create a locale in which every character is printable. In this case, a control sequence causing a line to be sent to the system and then executed can be sent to an unsuspecting user's terminal.

This is a security gap. Since most `talk` implementations involve transmitting each character to another process on the other end, in particular one run by the recipient's terminal, could lines 4171–4172: "typing characters from LC_CTYPE classifications `print` or `space` shall cause those characters to be sent to the terminal of the recipient," be interpreted as referring to the recipient's LC_CTYPE classification to close this gap?

If this solution is not possible, could "however, a privilege of the user may further constrain the domain of accessibility of the terminals of other users" be used to close this gap by disallowing users from talking to the terminals of other users if their LC_CTYPE is not a public locale?

Interpretation for IEEE Std 1003.2-1992

The description of LC_CTYPE on page 608, lines 4222–4223, makes the operation of `talk` undefined if the sender's and receiver's locales are not the same. This allows specific implementations of `talk` to prevent the security gap by disallowing talk when dissimilar locales are used. See also Interpretation #26.

Rationale for interpretation

None.

Interpretation Number: **23**
Topic: `test`
Relevant Clauses: 4.62.4
Classification: Unaddressed issue

Interpretation request

The standard describes, in 4.62.4, the `-w` and `-x` primaries:

> `-w` True if *file* exists and is writable. True shall indicate only that the write flag is on. The file shall not be writable on a read-only file system even if this test indicates true.
>
> `-x` True if *file* exists and is executable. True shall indicate only that the execute flag is on. If *file* is a directory, true indicates that *file* can be searched.

However, for the `-r` primary, the standard states that `-r` file returns "true if *file* exists and is readable." Does the `-r` primary have the same caveat as the `-w` and `-x` primaries, namely, only checking the *read* flag in the mode bits?

The descriptions of the `-w` and `-x` primaries mention the write flag and the execute flag. Which of the respective flags should be checked: owner, group, or other? Or is this unspecified by POSIX.2? Otherwise, does the standard intend that the appropriate flag be used: the user flag if the user ID matches the owner of the file, the group flag if the file is owned by any group in the current grouplist, or the other flag, with *no* check for appropriate privileges?

Interpretation for IEEE Std 1003.2-1992

The standard does not state that `-r` checks the read bit, so it does not have that restriction the way the `-w` and `-x` flags do. The question of which flags are checked (and the order) is unspecified by the standard, and thus any behavior is allowed. Concern over the wording of this subclause of the standard has been forwarded to the sponsor of the standard.

Rationale for interpretation

None.

Interpretation Number:	**24**
Topic:	`tr`
Relevant Clauses:	4.64.5.1
Classification:	Defect situation

Interpretation request

In 4.64.5.1, the standard states that the standard input to `tr` "can be any file type."

However, in 4.64.5.3, the standard states that the **LC_COLLATE** variable "shall determine the behavior of range expressions and equivalence classes" and in 4.64.7, the standard states that the `\octal` construct

> . . . can be used to represent characters with specific coded values. An octal sequence shall consist of a backslash followed by the longest sequence of one-, two-, or three-octal-digit characters (01234567). The sequence shall cause the character whose encoding is represented by the one-, two-, or three-digit octal integer to be placed into the array.

These two statements cause `tr` to be unusable on any files of type other than text. Historically, `tr` has been used to manipulate files containing binary data. For example, the perfectly valid and useful construct:

```
tr -d '\200-\2ff'
```

to delete all characters with the top bit on or even

```
tr '\200-\2ff' '\0-\1ff'
```

to strip the top bit (which are useful operations on binary files) no longer work.

For example, in the PC character set, `\200` is a C-cedilla, and `\2ff` is not defined as a glyph. Therefore, according to 4.64.5.3, the most likely interpretation is characters that collate from C-cedilla (probably the letter D) through the end will all match here. This is clearly wrong, not historical practice, and of no use whatsoever.

May the standard be interpreted as permitting octal escape sequences as endpoints of a range not to use the collating order, but rather byte ordering?

Interpretation for IEEE Std 1003.2-1992

The standard is clear in its requirement that octal sequences used as endpoints in a range be treated as collating elements. The implementation must follow this requirement. Concern over the wording of this area of this standard has been forwarded to the sponsor.

Rationale for interpretation

None.

Interpretation Number: **25**
Topic: `tr`
Relevant Clauses: 4.64.3, 4.64.7
Classification: Defect situation

Interpretation request

In 4.64.3, the standard states that the `-c` option means "complement the set of characters specified by *string1*. See 4.64.7" and in 4.64.7, the standard states that *c-c* "represents the range of collating elements between the range endpoints, inclusive, as defined by the current setting of the LC_COLLATE locale category."

In 4.64.7, the standard states that (lines 10549–10553)

> if the `-c` option is specified, the complement of the characters specified by *string1*—the set of all characters in the current character set, as defined by the setting of LC_CTYPE, except for those actually specified in the *string1* operand—shall be placed in the array in ascending collation sequence, as defined by the current setting of LC_COLLATE.

However, if the character set is ISO 646, for example, then the command

```
tr -c '[:print:]' '?'
```

which should translate all unprintable characters to question-mark characters, will pass bytes with the high bit set through unchanged. This is clearly wrong, not historical practice, and violates the principle of least astonishment.

The `tr` utility is a *binary* file manipulator. May the wording of lines 10549–10553 be interpreted as

> If the `-c` option is specified, the complement of the characters specified by *string1*—the set of all possible machine byte patterns, except for those actually specified in the *string1* operand—shall be placed in the array in ascending collation sequence, as defined by the current setting of LC_COLLATE.

to fix this problem?

Interpretation for IEEE Std 1003.2-1992

The standard is clear in its requirement that the `-c` option in this case will place the complement of the characters specified in *string1* in the array.

It is also clear that the definition of the complement is "the set of all current characters in the current character set, except for those actually specified in *string1*." This precludes an interpretation allowing the set of all possible machine byte patterns to be added to the complement set.

The implementation must follow these requirements. Concern over the wording of this area of this standard has been forwarded to the sponsor.

Rationale for interpretation

None.

Interpretation Number:	**26**
Topic:	`write`
Relevant Clauses:	5.37.2
Classification:	Defect situation

Interpretation request

In 5.37.2, the standard states that "typing other nonprintable characters shall cause implementation-defined sequences of printable characters to be written to the terminal of the recipient" and that "typing characters from LC_CTYPE classifications `print` or `space` shall cause those characters to be sent to the terminal of the recipient."

If {POSIX2_LOCALEDEF} is defined, a malicious user could create a locale in which every character is printable. In this case, a control sequence causing a line to be sent to the system and then executed can be sent to an unsuspecting user's terminal.

This is a security gap. Could lines 5985–5986: "however, a privilege of a user may further constrain the domain of accessibility of terminals of other users" be used to close this gap by disallowing users from writing to the terminals of other users if their LC_CTYPE is not a public locale?

Interpretation for IEEE Std 1003.2-1992

The standard allows the behavior described in the interpretation request. Concern over this has been forwarded to the sponsor of the standard.

Rationale for interpretation

None.

Interpretation Number: **28**
Topic: C binding for `execute` command
Relevant Clauses: E.9.3.1
Classification: Defect situation

Interpretation request

The specification for *system()* says that SIGCHLD is to be blocked (in the parent process) for the duration of the function. This is intended to prevent interception of a generated SIGCHLD due to the death of the created process before the *waitpid()* can get its status. This is beneficial, but it does not cover another circumstance: when SIGCHLD has been set to SIG_IGN on many systems.

On these systems, when the caller has set SIGCHLD to be ignored, no death of a child signal is generated and the *waitpid()* of the *system()* will return -1 once all children are finished and have set *errno* to ECHILD. Since it is quite likely that the command worked, it is unfortunate that the caller cannot tell.

Since IEEE Std 1003.2-1992 explicitly says that SIGCHLD is blocked, the caller that has set SIGCHLD to be caught can distinguish between a *system()* that temporarily resets SIGCHLD to SIG_DFL and one that temporarily blocks it, since there is a signal delivered in the second case.

To ensure that an application that sets SIGCHLD to SIG_IGN does not receive "bad" behavior from *system()*, it would be better to implement *system()* along these lines:

(1) Set SIGINT and SIGQUIT to SIG_IGN, noting the previous settings.

(2) Block SIGCHLD.

(3) Inquire about the disposition of SIGCHLD. If currently SIG_IGN, reset it to SIG_DFL.

(4) *fork()* child:

  (a) Reset SIGINT and SIGQUIT (and SIGCHLD if touched in item 3.)

  (b) Unblock SIGCHLD.

  (c) `exec` the shell command string.

  (d) *_exit(127)*. (The `exec` must have failed.)

(5) (`parent`) *waitpid()*, looping only on EINTR.

(6) Reset SIGINT and SIGQUIT (and SIGCHLD if touched in item 3.)

(7) Unblock SIGCHLD.

(8) Return the status of the child.

All child processes produced by the `exec`'d shell command will have SIGCHLD as the shell, and the commands will have a range of choices, as in the current edition of the standard. If SIGCHLD had a handler, the signal will be delivered just before the return, as it has been blocked until that point. Again, this matches the standard. For the duration of the function, SIGCHLD is reset to SIG_DFL only if it came in set to SIG_IGN.

This will otherwise behave in a similar manner to what is currently defined in the standard as far as any caller can tell.

The following changes should be made to IEEE Std 1003.2-1992:

(1) Page 1037—Add the following paragraph after line 11772:

If the setting of SIGCHLD to SIG_IGN precludes *waitpid()* from returning the status of any particular child, as is the case in System V, then after blocking SIGCHLD, if SIGCHLD is set to SIG_IGN, SIGCHLD should be temporarily reset to SIG_DFL. This is not necessary for those implementations in which *waitpid()* is unaffected by the setting of SIGCHLD. For such implementations, lines <*new_lines_added_below*> in the sample *system()* implementation shown in Figure E-8 would not be necessary.

(2) Page 1038—Change line 11805 to

```
struct sigaction sa, savintr, savequit, savechld;
```

(3) Page 1038—Add the following after line 11817:

```
sigaction(SIGCHLD, (struct sigaction *)0, &savechld);
if (savechld.sa_handler == SIG_IGN) {
   sa.sa_handler = SIGDFL;
   sigaction(SIGCHLD, &sa, (struct sigaction *)0);
   }
```

(4) Page 1038—Add the following after line 11820 and after line 11836:

```
if (savechld.sa_handler == SIG_IGN)
  sigaction(SIGCHLD, &savechld, (struct sigaction *)0);
```

Interpretation for IEEE Std 1003.2-1992

While the rationale does indeed say what is described by the requester, the normative text description of the behavior (B.3.1.2) overrides the description in the rationale and is the expected behavior. Concerns about this wording will be forwarded to the sponsor of the standard.

Rationale for interpretation

None.

Interpretation Number:	**29**
Topic:	Regular expressions
Relevant Clauses:	2.8
Classification:	No change required

Interpretation request

Please provide an interpretation of the following taken from 2.8 of IEEE Std 1003.2-1992.

Given a locale in which "ch" is a multiple character collating element that collates between "c" and "d," then certainly

[[.ch.]] matches "ch"

This makes it clear that

[^[.ch.]] doesn't match "ch" (and not even just the "c")

Therefore, consistency argues that

[^c] matches "ch"

And, of course,

[c] doesn't match "ch" (and not even just the "c")

If this is correct, then the simple rule is that if the string to check against a bracket expression can be taken as a multiple character collating element, then the matching process must do so.

In addition, what would be the behavior for character classes? Take, for example,

[[:alpha:]]

when presented with "ch." The rationale for IEEE Std 1003.2-1992 confirms that "character classes are not intended to include collating elements." However, there are still two possible answers: "ch" does not match, and the "c" of "ch" matches. Neither of these answers is helpful and neither shows that "ch" should match as a unit. Even worse, the nonportable [a-z] *does* match the unit "ch"!

What is actually specified for [[:alpha:]] here?

Interpretation for IEEE Std 1003.2-1992

A character class expression is defined in 2.8.3.2 as a set of characters belonging to a character class, as defined in the LC_CTYPE category of the current locale. A range expression is defined in the same subclause as a set of collating elements that fall between two elements in the current collation sequence, inclusive.

Thus, a collating element "ch," which is not a character, would be matched by the range expression [a-z], but not by the character class (set of specific characters specified in the locale file) [:alpha:]. The character class [:alpha:] would match the "c" and the "h" individually, for the same reason that the expression [c] matches the "c" in ch, but not the collating element "ch."

Rationale for interpretation

None.

Interpretation Number:	**30**
Topic:	`glob`
Relevant Clauses:	B.8.2
Classification:	Ambiguous

Interpretation request

In B.8.2, the standard states:

> The argument *pattern* is a pointer to a pathname pattern to be expanded. The *glob()* function shall match all accessible pathnames against this pattern and develop a list of all pathnames that match.

This is clearly the appropriate behavior when *pattern* contains at least one unescaped metacharacter, but what if *pattern* is a simple string that does not match an existing accessible pathname? If taken at its face value, one might expect *glob()* to return GLOB_NOMATCH, but this is not the behavior of the shell in the same situation. In particular, the shell only checks for matches of a command argument against accessible pathnames when there is at least one metacharacter in the argument.

The intent for *glob()* was to provide the shell's argument expansion through a simple API. If this is correct, only the presence of metacharacters will potentially cause *glob()* to return GLOB_NOMATCH. If not, then an application would potentially be forced into scanning the pattern string on failure to determine whether a pattern replacement (due to a metacharacter) was at fault or whether a simple string did not match an existing pathname; such scanning would have to take escaping into account, as well as any potential "extension" metacharacters, so the process is not simple.

A clarification of the intended behavior in this case is requested.

Interpretation for IEEE Std 1003.2-1992

The standard is specific in its specification that this is the pattern to be matched against, and that `pglob->gl_pathc` shall be zero if GLOB_NOCHECK is clear and there is no match. However, since it is not clearly stated that nonmatch is a "failure," it is not clear that GLOB_NOMATCH is required to be returned. Therefore, *glob()* may return either zero or GLOB_NOMATCH. Concern over wording of this area of the standard is being forwarded to the sponsor of the standard.

Rationale for interpretation

None.