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, December 1994 Edition

IEEE

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

December 30, 1994 *SH94253*

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

IEEE Std 1003.2-1992/INT
December 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

Copyright © 1994 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1994. Printed in the United States of America.

ISBN 1-55937-476-4

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

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

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.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

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.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Introduction

After approval and publication of a standard, questions may arise regarding the meaning of portions of a standards as it relates to its users. 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 November 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:	Lowell Johnson
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

Rachel A. Meisel
IEEE Standards Project Editor

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**31**
Topic:	`ex`—Replacement strings
Relevant Clauses:	5.10.7.4
Classification:	Defect

Interpretation request

Historically, the upper and lower case manipulations occurred on all characters in the replacement strings, not just those inserted using `\n` or `\[1-9]`. Was it the intent of the standard to change historical practice? If not, the following wording is suggested:

> The strings `\l`, `\u`, `\L`, and `\U` can be used to modify the case of elements in the replacement string. The string `\l` (`\u`) shall cause the character that follows to be converted to lowercase (uppercase). The strings `\L` (`\U`) shall cause all characters subsequent to them to be converted to lowercase (uppercase) until the string `\e` or `\E`, or the end of the replacement string, is encountered.

Interpretation for IEEE Std 1003.2-1992

The standard is specific in its requirement that `\l`, `\u` ,`\L`, and `\U` should only modify the case of elements using the & and `\digit` notation.

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

Rationale for interpretation

None.

Editorial note for future revision of standard (not part of this interpretation)

Consider applying the above proposed change to 5.10.7.4, Replacement Strings.

NOTE—There is no certainty that these comments will apply to any revision of IEEE Std 1003.2-1992.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**3 2**
Topic:	`ed`—List command
Relevant Clauses:	4.20.7.3.14
Classification:	Defect

Interpretation request

The newline (`\n`) character is listed in Table 2-16. Historically, the *list* command within `ed` did not display the newline terminating each line in a special mode. Was it the intent of the standard to change this?

Interpretation for IEEE Std 1003.2-1992

The standard states that `ed` should display the newline terminating each line in a special mode for the list command and conforming implementations shall conform to this.

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

Rationale for interpretation

None.

Editorial note for future revision of standard (not part of this interpretation)

A proposed change to 4.20.7.3.14, List Command, is to replace the second sentence, starting on line 3825, with the following:

> The characters listed in Table 2-16 (see 2.12), except for `\n`, shall be written as the corresponding escape sequences.

NOTE—There is no certainty that these comments will apply to any revision of IEEE Std 1003.2-1992.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**3 3**
Topic:	`sed`—Extended description
Relevant Clauses:	4.55.7
Classification:	Ambiguous

Interpretation request

Was it the intent of the standard to use *command* both in line 8629 and again in the subsequent text (particularly in line 8803)? The two uses were not the same in historic implementations of `sed`.

Interpretation for IEEE Std 1003.2-1992

The first part of 4.55.7 describes the form of a command using the term *command*, thereby making the meaning of *command* in 4.55.7 ambiguous.

The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**3 4**
Topic:	sed REs
Relevant Clauses:	4.55.7.2
Classification:	Unaddressed issue

Interpretation request

Was it the intent of the standard to ignore empty REs? Historic implementations of sed used the empty RE (//) to match the last RE used.

Interpretation for IEEE Std 1003.2-1992

The meaning of an empty RE is not defined by the standard.

The standard does not speak to the meaning of a null BRE for sed. The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

Rationale for interpretation

None.

Editorial note for future revision of standard (not part of this interpretation)

A proposed change to 4.55.7.2, sed REs, is to add the following paragraph (3):

> (3) If an RE is empty (i.e., no pattern is specified) sed shall behave as if the last RE used in the last command applied (either as an address or as part of a substitute command) was specified.

NOTE—There is no certainty that these comments will apply to any revision of IEEE Std 1003.2-1992.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**35.1**
Topic:	`sed` editing commands
Relevant Clauses:	4.55.7.3
Classification:	Unaddressed issues

Interpretation request

(1) Was it the intent of the standard to not describe that multiple commands may be entered on a single line by interspersing the commands with semi-colons?

(2) Page 412, lines 8784–8785: Does the standard require white space before the *wfile* argument to the w flag to the substitute command?

(3) Page 413, lines 8794–8802: Was it the intent of the standard to ignore that the `y` command has historically used the string `\n` to cause `sed` to insert a newline into the string?

(4) Page 413, lines 8803–8806: It is unclear in the standard what happens when the *command* is itself the ! character, i.e., what if the user enters

[2addr]!!!!!

Interpretation for IEEE Std 1003.2-1992

(1) The purpose of the interpretation process is not to determine intent. Using a semi-colon character (`;`) in `sed` to separate commands would be a legitimate extension. However, in a few cases, (for example, the w command) the standard would require that the semi-colon character be interpreted as part of *wfile* unless there was white space between *wfile* and the semi-colon character.

The standard is unclear on these issues, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(2) The standard is unclear on these issues, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(3) The standard is unclear on these issues, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(4) The behavior of [2addr]!!!!! is unspecified. The standard is unclear on these issues, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**35.2**
Topic:	`sed` editing commands
Relevant Clauses:	4.55.7.3
Classification:	Defects

Interpretation request

(1) Page 410, lines 8700–8703: Was it the intention of the standard to change historical practice by not including `n` as a command that can cause the appended text to be written to standard output?

(2) Page 411, lines 8727–8734: The newline character is listed in Table 2-16. Was it the intent of the standard to change historical practice by having `sed` display the newline terminating each line in a special format?

(3) Page 412, lines 8767–8770: Was it the intent of the standard to change historic practice by not discarding the backslashes from the replacement string?

Interpretation for IEEE Std 1003.2-1992

(1) The standard states that the `N` command only is acceptable in this case, and conforming implementations shall conform to this.

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

(2) The standard states that `sed` shall display the newline terminating each line in a special format, and conforming implementations shall conform to this.

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

(3) The standard states on page 412, line 8767, that

> "For each backslash (\) encountered in scanning *replacement* from beginning to end, the following character shall lose its special meaning (if any)."

and conforming implementations shall conform to this.

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

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**35.3**
Topic:	`sed` editing commands
Relevant Clauses:	4.55.7.3
Classification:	Ambiguous

Interpretation request

On pages 411, lines 8739–8744, was it the intent of the standard to change historical practice by not having the `n` and `N` commands branch to the end of the script if they encounter the end of the input?

Interpretation for IEEE Std 1003.2-1992

The standard is unclear regarding the specific behavior of the `sed` commands `n` and `N` when the end of input is reached while in a script, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

Rationale for interpretation

None.

IEEE

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**3 6**
Topic:	`ed` commands and extended description
Relevant Clauses:	4.20.7.3, 5.10.7, 5.35.7
Classification:	Ambiguous

Interpretation request

The standard does not seem to describe what behavior is required from the three editors `ed`, `ex`, and `vi` when an end-of-file is encountered during either command input or text input. Please clarify what the utility implementations are expected to do.

Historic practice appears to be both inconsistent and wrong. It is strongly believed that the commands should behave very similarly to what happens when a SIGHUP is received, i.e., save the user's work so that it can be recovered, and exit with an error status.

Interpretation for IEEE Std 1003.2-1992

The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

Rationale for interpretation

None.

Editorial note for future revision of standard (not part of this interpretation)

A proposed change to 4.20.7.3, `ed` Commands, is to replace the sixth paragraph (lines 3690–3691) with the following:

> If an end-of-file is detected on standard input when input from the user is expected and the standard input is a terminal device, if the buffer is not empty and has changed since the last write, the `ed` utility shall attempt to write a copy of the buffer to a file. First, the file named `ed.hup` in the current directory shall be used; if that fails, the file named `ed.hup` in the directory named by the **HOME** environment variable shall be used.
>
> If an end-of-file is detected on standard input when input from the user is expected and standard input is not a terminal device, the `ed` utility shall not write the file to the currently remembered pathname or return to command mode, and shall terminate with a nonzero exit status.

NOTE—There is no certainty that these comments will apply to any revision of IEEE Std 1003.2-1992.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**37**
Topic:	`more`—Input files
Relevant Clauses:	5.18.5.2
Classification:	Defect

Interpretation request

Was it the intent of the standard to disallow the historic practice of using `/dev/tty` as an input device if standard error is not readable?

Interpretation for IEEE Std 1003.2-1992

The standard states on lines 2807–2810 that if standard output is a terminal, standard error is not readable, and command input is needed and `more` shall terminate with an error indicating that it was unable to read user commands. Conforming implementations shall conform to this.

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

Rationale for interpretation

None.

Editorial note for future revision of standard (not part of this interpretation)

A proposed change to 5.18.5.2, Input Files, is to replace the text of the subclause with the following:

> The input files being examined shall be text files. If the standard output is a terminal, standard error shall be used to read commands from the user. If the standard output is a terminal, standard error is not readable, and command input is needed, `more` may attempt to obtain user commands from the controlling terminal (e.g., `/dev/tty`); otherwise, `more` shall terminate with an error indicating it was unable to read user commands. If standard output is not a terminal, no error shall result if standard error cannot be opened for reading.

NOTE—There is no certainty that these comments will apply to any revision of IEEE Std 1003.2-1992.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**38**
Topic:	`ex`—Extended description
Relevant Clauses:	5.10.7
Classification:	Defect

Interpretation request

The specification that “no `.exrc` file shall be read unless it is owned by the same user ID as the effective user ID of the process” is necessary but not sufficient. To keep the `.exrc` files from being a security problem, the file should not be read if it is writable by anyone other than the owner.

Interpretation for IEEE Std 1003.2-1992

The standard states the required behavior and conforming implementations shall conform to this.

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

Rationale for interpretation

Interpretations cannot make substantive changes to the standard. This may be considered for a future revision.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**39**
Topic:	`ls`
Relevant Clauses:	4.39.8
Classification:	No change

Interpretation request

Line 6071 states that `ls` should exit with 0 if “all files were written successfully.” The `ls` command does not normally write files, but this implies that it does.

A clarification of the intended behavior is required.

Interpretation for IEEE Std 1003.2-1992

The sentence means the names of all relevant files were written successfully.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**40**
Topic:	Internationalization issues
Relevant Clauses:	2.5.2.2
Classification:	Ambiguous

Interpretation request

POSIX.2 has defined a mechanism for talking about multi-character sequences as a single unit, namely as collating elements (CEs). Although CEs are motivated by sorting issues, they appear in REs. This leads to the question of how to parse text into CEs. There are many possible answers, and furthermore, the parsing might be affected by context. For example, given the usual alphabet augmented by the collating element `<ij>` defined as `<i><j>`, can the string `ij` ever be parsed as two collating elements?

Subclause 2.5.2.2, line 1668, says in the context of sorting, "strings are first broken up into a series of collating elements." Does this apply to pattern matching? And if so, how exactly is this done (for sorting or pattern matching)?

A proposed solution is to add the following text somewhere within 2.5.2.2. This text should be referred to by line 1668 and by the general RE introduction (2.8.2).

> When a string is interpreted as a sequence of CEs, the sequence shall be as found by the following process:
>
>> Starting at the first character of the string, determine the longest prefix of the string that matches a CE, add that CE to the sequence and continue this process with the character after that prefix until the string is exhausted.

Note that this applies even if a sort key indicates that a piece of the text is processed in backwards (right-to-left) order; that is, the right-to-left processing applies to the CEs found by a left-to-right lexical scan.

Rationale for interpretation

This is the greedy algorithm normally done in lexical analysis. Any other choice would require backtracking with potentially exponential runtime. It implies that, when `<i><j>` is a collating element, under no circumstances can a bracket expression match the *i* alone in the string *ij*. In particular, neither `[[.i.][.ij.]]j` nor `[[.i.]]j` matches *ij*. By contrast, `i[[.j.]]` does match *ij*, because in this regular expression *i* denotes a character and is unaffected by concerns about collating elements.

Interpretation for IEEE Std 1003.2-1992

The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**41**
Topic:	Internationalization issues—Locales
Relevant Clauses:	2.5.2.1, 2.8.3.2 (6)
Classification:	1–6: Unaddressed issues 7: Ambiguous issue 8: No change

Interpretation request

(1) The specification of locales and the interface to them discriminates against nonvendor-supplied software. In particular, it is impossible to write a portable implementation of *regcomp*() and *regexec*(), as there is no standardized interface to the vital knowledge presumably set up by a call to *setlocale*(). This knowledge is detailed below. In brief, the first seems an oversight and the others are necessary to use the locale information.

(2) How can membership in class `[:blank:]` be determined portably?

A proposed solution is to provide a *ctype* function *isblank*().

Rationale—It is inconsistent that this be the only LC_CTYPE category without a C binding. Note that this extension introduces a difference between the C and POSIX locales.

(3) How can the meaning of an arbitrary equivalence class be discovered portably?

A proposed solution is to provide a function that, given any name for an equivalence class, returns a list of names of collating symbols in the class. The order of the list shall be the same regardless of what name is given.

Rationale—This is needed if an application, such as a searching or sorting tool, requires this locale-specific information. In particular *regcomp*() and `sort` need it.

(4) How can the meaning/value of an arbitrary collating symbol be determined portably?

A proposed solution is to provide a function that, given a collating symbol, returns the representation and length of the symbol.

Rationale—This is needed if an application, such as a searching or sorting tool, requires this locale-specific information.

(5) How can the collating elements in a string be found and compared portably?

A proposed solution is to provide a function that returns the length and the weight vector for the collating element at the beginning of the string.

Rationale—This is needed if an application, such as a searching or sorting tool, requires this locale-specific information.

(6) How can *regcomp*() expand a range expression into a list of collating elements portably?

A proposed solution is to provide a successor function that, given the name of a collating element, returns the name of the collating element with the next larger weight vector. For this purpose two elements with the same weight vector compare in the order of their equivalence listing.

Rationale—This is needed if an application, such as a searching or sorting tool, requires this locale-specific information. It may further be useful to have a way to inquire whether a locale contains any multi-character collating elements.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

(7) Lines 2918–2920 say that an equivalence class expression that names a collating element not in an equivalence class shall be treated as a collating symbol. Does this statement affect the meaning of `collating_symbol` in line 3306? Does it eliminate such equivalence class expressions from consideration in lines 2943–2945?

A proposed solution is to change lines 2918–2920 to say "the expression shall be understood as an equivalence class that contains only the one collating element."

Actually, the admittance of singleton equivalence classes in the definitions of 2.5.2.2 would be preferred.

Rationale—This question affects the meaning of range expressions. Lines 2918–2920 could be construed as forcing `[[=CE1=]-[=CE2=]]` to mean `[[.CE1.]-[.CE2.]]` in some cases, although the former expression looks syntactically incorrect. The preferred solution agrees with customary mathematical usage, and clarifies the behavior of the equivalence-class function proposed in (3) above.

(8) What if collation changes between *regcomp*() and *regexec*()?

A proposed solution is that the result is undefined.

Rationale—For the common case of locales in which all collating elements are single characters, *regcomp*() should be allowed to compile character classes. At the same time, *regexec*() should be allowed to handle multi-character collating symbols. The proposed resolution assures that both desiderata are met.

Interpretation for IEEE Std 1003.2-1992

(1) The standard does not speak to this issue and no conformance distinction can be made between alternative implementations based on this.

The standard does not require that an implementation conforming to the standard be portable. Therefore, there is no requirement that the functionality be specified by the standard.

Concerns are being forwarded to the sponsor.

(2–6) The standard does not speak to these issues and no conformance distinction can be made between alternative implementations based on this.

Concerns are being forwarded to the sponsor.

(7) The standard is unclear on this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(8) The standard states the required behavior and conforming implementations shall conform to this. According to B.5.2, lines 367–368 on page 729, the standard specifies the result is undefined.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number: **43.1**
Topic: Regular expressions—Imprecise specification
Relevant Clauses: 2.8
Classification: 1: Ambiguous
2: Defect

Interpretation request

These questions address areas of imprecision in the specifications of REs. With regard to question (2), 1003.2 uses various terms roughly synonymously (leftmost, first, earliest; subpattern, subexpression) for describing a left-to-right order across a line of text. It would be well if the text (and rationale) used one term consistently.

(1) Can a duplicated subexpression match the null string? If so, will the duplication be repeated until the expression does match the null string?

A proposed solution is a subexpression that can match a null string shall not be duplicated.

Rationale—Although adjacent duplication symbols are illegal (for no apparent reason, particularly for EREs), `\(x*\)*` is a legal expression, in which the `*`s are not adjacent. This raises the question: How many times is the null string matched?

Suppose that `\(x*\)*` were allowed. Does matching it to `xxx yield \1 = xxx or \1 = null`? The latter alternative is consistent with the rule that a null string is longer than no match. By extending `xxx` with a null string instead of nothing, one gets a longer match. More null strings make even longer matches.

To avert metaphysical questions about the last element of an infinite sequence, or an element following an infinite sequence, one could forbid null iterations. But this has an unsatisfying corollary that `\(x*\)*` would not match the null string. Compromise positions might forbid null iterations after the first iteration or after the first null iteration. Such special pleading and the resultant implementation complexity is not worth the return.

One possible implementation is a no-null-unless-first policy. Another implementation is a repeat-until-null policy. The latter implementation seems perhaps less strained, until the realization that it makes references (`\1`, `\2`, etc.) to contents of duplications useless because they will always be null or undefined.

The proposed solution also outlaws EREs such as

(a?|b)*

(a?)\{2,2\}

([a-z]*(|^))*

The set of regular languages is unchanged, however. There would still be legal equivalents to these and all other outlawed expressions.

An alternative, to leave the meaning undefined, is unacceptable. Users will be unaware of the exact bounds of portability, meaning that some complex `sed` scripts may not be portable.

(2) What is the meaning of the BRE `\(\)` (see 2.8.5.2)?

A proposed solution is to forbid this expression.

It would be also acceptable, but rather less so, to define that the pattern `\(\)` matches the null string.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Rationale—The pattern has no analog in EREs, no utility, and no basis in history. It should thus be banned. Otherwise, at least define what it means.

Interpretation for IEEE Std 1003.2-1992

(1) The standard does not require the successful matching of a null string after a successful match on a non-null string for duplication symbols. However, the standard does not clearly say that you are not allowed to match null strings after a successful match. Therefore, the standard is ambiguous.

The definition of a back reference expression in 2.8.3.3 does not specify which match of a subexpression followed by a duplication symbol is to be returned. Note, however, that the definition *regcomp*() defined in B.5.1, lines 344–346 on page 728, indicates that the back reference expression will match the last string matched by the subexpression.

The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(2) Subclause 2.8.5.2, line 3263 on page 89, requires this form to be accepted. But, the text in 2.8.3 does not describe its meaning.

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

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**44**
Topic:	Regular expression error codes
Relevant Clauses:	Table B-10
Classification:	Unaddressed issue

Interpretation request

Must the regular expression error codes be distinct?

A proposed solution is that the values in Table B-10, along with any other REG_ values defined by the implementation (line 430, page 730), must all be different.

Rationale—A conforming *regcomp*() could return REG_BADPAT for every syntax error. Nevertheless, it is important that the other codes be distinct, lest a C switch on the codes be illegal.

Interpretation for IEEE Std 1003.2-1992

The standard does not address this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**4 5**
Topic:	*pmatch*( ) rules
Relevant Clauses:	B.5.2
Classification:	Defect

Interpretation request

In B.5.2, the meaning of lines 356–361, on pages 728 and 729, is unclear.

A proposed solution is to replace the five rules with the following three:

(1) The nesting of substrings denoted by nonnegative offsets in *pmatch* shall be the same as the nesting of subexpressions in pattern.

(2) A substring denoted by nonnegative offsets in *pmatch*[*i*] shall be the last of all matches to subexpression *i*.

(3) The offsets in *pmatch*[*i*] shall be nonnegative if possible according to rules (1) and (2), otherwise the offsets in *pmatch*[*i*] shall be –1.

Rationale—One of us understood that "the match ... reported in *pmatch*[*i*] shall be ... within the substring ... rather than the whole string," to mean that the offsets in *pmatch*[*i*] were to be measured relative to an origin specified by *pmatch*[*j*]. It is now believed that the rule merely says that reported matches must nest the same way as the subexpressions. The proposed solution captures this idea succinctly. Rule (5) could be kept, it is not needed, given lines 289–290.

Interpretation for IEEE Std 1003.2-1992

The standard states the required behavior and conforming implementations shall conform to this.

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

Rationale for interpretation

Interpretations cannot make substantive changes to the standard. This may be considered for a future revision.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**4 6**
Topic:	Case construct
Relevant Clauses:	3.9.4.3
Classification:	Defect

Interpretation request

(1) In 3.9.4.3, the `case` construct appears to be described as requiring a minimum of two compound lists. This appears to contradict the intent of the standard to codify existing historical practice, which allows `case_item` to be omitted.

(2) Also in 3.9.4.3, the grammar appears to contradict historical practice by disallowing the omission of the final `;;`. Is this interpretation of the standard correct? If so, then the standard is in error on these points.

Interpretation for IEEE Std 1003.2-1992

(1) The standard is clear that it allows multiple compound lists and multiple patterns to exist, but does not require it.

(2) The wording in the standard is in conflict. For example, 3.9.4.3 lines 935–936 is in conflict with line 944. However, the standard states that the grammar shall prevail, requiring that the use of `;;` is required for conformance.

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

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**47**
Topic:	`od`
Relevant Clauses:	4.45.7
Classification:	Defect

Interpretation request

In 4.45.7, the last sentence of the first paragraph states

> If no output type is specified, the default output format shall be as if `-t o2` had been specified.

which indicates that 2 bytes are to be transformed by each instance of the (octal) output type.

This contradicts historical practice, in which the `od` command by default does an octal conversion on items of type *short*, regardless of byte or word size.

Is this interpretation of the standard correct? If so, then the standard is in error on these points.

Interpretation for IEEE Std 1003.2-1992

The standard states the required behavior and conforming implementations shall conform to this.

The effect of this is that a system that does not support 2 byte interval quantities could produce a diagnostic message. So, concerns about the wording of this part of the standard have been forwarded to the sponsor.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**49**
Topic:	`ex—autowrite, aw` option
Relevant Clauses:	5.10.7.5.3
Classification:	Defect

Interpretation request

Was it the intent of POSIX.2 to change historic practice by making the `edit` command be affected by the `autowrite` option?

Interpretation for IEEE Std 1003.2-1992

The standard states the behavior required and conforming implementations shall conform to this.

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

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**50**
Topic:	`ex`—`autowrite, aw` option
Relevant Clauses:	5.10.7.5.3
Classification:	Ambiguous

Interpretation request

Was it the intent of POSIX.2 to change historic practice by requiring that the file be written back if it had ever been modified, or to match historic practice of only writing the file back if it had been modified since it was last written out?

Interpretation for IEEE Std 1003.2-1992

The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

The standard does not define what modified means.

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

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number: **51**
Topic: `ex—beautify, bf` option
Relevant Clauses: 5.10.7.5.4
Classification: Defect

Interpretation request

Was it the intent of the POSIX.2 to change historic practice by requiring that `beautify` affect text read in from files?

Historic practice is as follows, if the `beautify` option is set.

In `ex` mode, keys that were not already specially handled, that were less than an ASCII space, or that were the del (0177) key, and were neither a tab nor a form-feed, and were read in from a COMMAND file, were discarded. When the first `^H` was discarded a message was written to the terminal. Quoting (a `\`) would keep the keys from being discarded.

In `vi` mode, keys that were not already specially handled, that were less than an ASCII space, or that were the del (0177) key, and were neither a tab nor a form-feed, and were entered in input mode (either to the edit buffer or to the colon command line), were discarded. Quoting (a `^V`) would keep the keys from being discarded.

Suggestion—Word it such that implementations are permitted to interpret different sets of special keys, and that any keys that are not quoted, which are not specially interpreted, and for which *iscntrl*() returns true, are discarded.

Interpretation for IEEE Std 1003.2-1992

The standard states the behavior required and conforming implementations shall conform to this.

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

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**52**
Topic:	`vi`—Escape next character
Relevant Clauses:	5.35.7.1.14
Classification:	No change.

Interpretation request

Was it the intent of POSIX.2 to change historic practice by requiring that `^Q` be used to escape the next character?

Historic practice is that it be used as a flow control character, and that it only be used as a literal next character if it is not already used for flow control.

Interpretation for IEEE Std 1003.2-1992

The rationale on page 1009, lines 10603–10609 shows the expected behavior if `control-Q` is currently being used as a flow control character. Its status as a control character overrides the wording in 5.35.7.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number: **53**
Topic: `xargs -e`*eofstr*—Default behavior
Relevant Clauses: 4.72.2, E.4.72
Classification: Unaddressed issue

Interpretation request

There is an inconsistency between the description of the `xargs` utility in 4.72.2 and historical behavior of the utility's `-e`*eofstr* option as described in E.4.72. It is requested that the next revision of the standard clarify the intent by changing either the description or the rationale.

Specifically, historic implementations of the `xargs` utility (as specified by the System V Interface Definition, Issue 3, and by the X/Open Portability Guide XSI Commands and Utilities, Issue 3) would stop processing when a line containing exactly one underscore character was encountered on standard input. The rationale for E.4.72, lines 8985–8987 on page 970, says that the `-e`*eofstr* option was not included in the standard because this feature could easily be replaced by features provided by the `sed` utility. If the only use of the `-e`*eofstr* option were to specify a line other than a line containing exactly one underscore character as the end-of-file condition and the POSIX.2 description of the `xargs` utility specified that a line containing exactly one underscore character was to be interpreted as an end-of-file condition, this would be true. However, the POSIX.2 description of the `xargs` utility (see 4.72.2, lines 11100–11124 on page 480,) does not allow for any way to specify an input line to be treated as an end-of-file condition and the historical implementation's `-e`*eofstr* option could be used not only to change the end-of-file string, but also to disable end-of-file string processing.

When the standard behavior broke historic application practice, the rationale in POSIX.2 usually pointed out the changed behavior. Since it does not mention this case, it is believed that the breakage was unintentional. Although it is understood that the normative text in 4.47.2 overrides the informative rationale in E.4.47, it is hoped that the interpretations committee will consider the intent and suggest that this breakage of historic practice be remedied in the next revision of the standard.

Interpretation for IEEE Std 1003.2-1992

The standard does not speak to this issue and hence does not preclude implementations supporting this option.

Rationale for interpretation

Interpretations are not intended to constitute an alteration to the standard.

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

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**54**
Topic:	`expr`
Relevant Clauses:	4.22
Classification:	Ambiguous

Interpretation request

If no operands are provided, i.e., there is no expression, then is the expression invalid?

Some argue that no expression is tantamount to an invalid expression (exit status 2). Others argue that no expression falls under "an error occurred" (exit status >2). What is the correct interpretation of the standard or are both interpretations correct?

Note that both exit status 2 and 3 from different implementations of `expr` have been seen.

Interpretation for IEEE Std 1003.2-1992

The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this.

The standard specifies how `expr` should act if given one or more operands, but the standard does not address the issue when no operands are given.

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

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**55**
Topic:	`ex—errorbells, eb` option
Relevant Clauses:	5.10.7.5.5
Classification:	Ambiguous

Interpretation request

There are things in the specification of the `ex/vi errorbells` option that differ from historic practice.

(1) Historically, the open/visual modes were not affected by the `errorbells` option. Is there any discussion of how `vi` and `ex` relate in terms of the various `ex` options that `vi` can set?

(2) It was not historic practice to precede error messages with an alert action if `errorbells` was set, as the first sentence of the paragraph requires. The historic behavior was that if the editor was in `ex` mode, and there was no standout mode capability for the terminal, and `errorbells` was set, error messages were announced by an alert character.

It seems as if the intent of the standard is correct, but the last sentence in the paragraph appears to contradict the first one. The second sentence may not actually be English. Is the standard trying to discuss the visual bell capabilities of `vi`, but the `errorbells` option did not affect that?

Was it the intent of POSIX.2 to change historic practice in these ways?

Also, there is a typographical error on page 537, line 1868 ("the the").

Interpretation for IEEE Std 1003.2-1992

The wording in the sentence beginning on page 537 line 1864 along with the last sentence starting on line 1870 are in conflict and therefore the action to be taken by an implementation when the `errorbells, eb` option is set is unclear.

The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**5 6**
Topic:	`substitute`
Relevant Sections:	5.10.7.2.30
Classification:	1: Ambiguous 2–5: Defect 6: Unaddressed

Interpretation request

There are things in the specification of the `ex/vi substitute` command that differ from historic practice.

(1) There is no wording that limits the option characters that are accepted by implementations. Historically, the options were limited to `c`, `g`, and `r`.

(2) Historic implementations accepted the `r` option as well as the `c` and `g` options. The effect of the `r` option was to cause the use of the last BRE used in any command, the same as the ~ command.

(3) Historically, the `c` and `g` options were toggled, e.g., the command `:s/abc/def/` was the same as `s/abc/def/ccccgggg`.

(4) Historically, any substitute could be interrupted by SIGINT, not just those having the `c` option.

(5) Historically, was the *substitute* command affected by the *wrapscan* option?

(6) The historic `edcompatible` option is missing from the standard. The behavior of the `edcompatible` option was that it made the values of the `c` and `g` suffices remembered instead of being reinitialized to "off" for each substitute command. The single special case was that they were always reinitialized to 0 if the pattern and replacement strings were specified.

Was it the intent of POSIX.2 to change historic practice in these ways?

One other comment. The current wording of 5.10.7.2.30 requires `vi` mode to implement the historic practice of displaying the line with ^ characters under it, something that is incredibly inefficient and antiquated given the capabilities of current terminals. It would be nice to permit superior `vi` implementations that, for example, highlighted the characters to be substituted instead of redisplaying the line and destroying the screen presentation. (There are at least two implementations of `vi` that have this capability.)

Interpretation for IEEE Std 1003.2-1992

(1–2) The standard clearly states behavior for the `c` and `g` options, and conforming implementations shall conform to this.

For the `r` option the standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. The `c` and `g` options are only the minimum requirements, implementations may provide additional facilities. This is being referred to the sponsor.

(3) The standard states behavior for the `c` and `g` options, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

(4) The standard states behavior for the `c` option and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(5) The standard states the behavior for the interaction with the `wrapscan` option, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(6) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

Last paragraph:

The request is substantially identical to interpretation number 63, item 1, in this document, and the resolution of that interpretation applies in this case.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**57**
Topic:	`vi`—Terminate command or input mode
Relevant Clauses:	5.35.7.1.17
Classification	No change

Interpretation request

The current `vi` specification requires that the key sequence `[0-9]*<ESC>` be treated as an error. The historic `vi` has no way for users to erase partial commands that consist only of the preliminary number.

It would be helpful if the standard permitted the `<ESC>` key to cancel partial commands that only consist of the preliminary number. This is consistent with historic practice, and provides a superior implementation. There is at least one implementation of `vi` that behaves this way.

Was it the intent of POSIX.2 to disallow a superior implementation of this functionality?

Interpretation for IEEE Std 1003.2-1992

The standard clearly states the behavior for `<ESC>` and conforming implementations shall conform to this.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**58**
Topic:	`find` time primaries
Relevant Clauses:	4.24.4
Classification:	Ambiguous

Interpretation request

An interpretation of the description of the `find` utility in 4.24.4 in the cases described below is requested.

The descriptions of the `find` utility's `-atime` *n* (on page 280, lines 4252–4257), `-ctime` *n* (on page 280, lines 4258–4263), and `-mtime` *n* (on page 281, lines 4291–4296) primaries work OK when the *n* option argument is not preceded by plus or minus sign.

When the *n* option argument is preceded by a minus sign, the descriptions listed above along with the wording on page 280, line 4250 say that the appropriate time subtracted from the initialization time is less than *n*–1 to *n* multiples of 24 h. Does this mean that the time is less than *n*–1 multiples of 24 h, less than *n* multiples of 24 h, or something else?

When the *n* option argument is preceded by a plus sign, the descriptions listed above along with the wording on page 280, line 4248 say that the appropriate time subtracted from the initialization time is greater than *n*–1 to *n* multiples of 24 h. Does this mean that the time is greater than *n*–1 multiples of 24 h, greater than *n* multiples of 24 h, or something else?

The rationale for 4.24.4 found in E.4.24, on page 906–907, lines 6306-6316, indicates that the intent was to document existing practice. It looks like the standard lost clarity when it converted the concept of days to an *n* option argument using the same description for this concept as is used for the `-links` *n* and `-size` *n*[c].

Interpretation for IEEE Std 1003.2-1992

The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**59**
Topic:	`csplit` operands
Relevant Clauses:	5.6.4
Classification	Defect

Interpretation request

An interpretation of the behavior of csplit as outlined in 5.6.4 is requested.

The description of the use of + or – as described on page 503, lines 608 and 609 breaks historical practice. What was the reason for deviating from historical practice?

The description of which line should be used as the current line in performing a pattern match of `rexp` breaks historical practice. What was the reason for deviating from historical practice?

Interpretation for IEEE Std 1003.2-1992

The standard states the behavior for + and –, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**6 0**
Topic:	Regular expressions (REs), from IEEE Std 1003.2-1992/INT, March 1994 Edition, number 2
Relevant Clauses:	B.5.2
Classification:	Defect

Interpretation request

Interpretation number 2 in IEEE Std 1003.2-1992/INT, March 1994 Edition appears to be an incorrect change in terms of the meaning of the words in POSIX.2. Could the IEEE clarify the situation?

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.

POSIX.2 is clear on page 727, line 285, that *re_nsub* contains the number of PARENTHESIZED subexpressions, which is different from the total number of subexpressions because *pattern* itself counts as a subexpression (see page 728, line 337).

The interpretation given adds one to the value stored in *re_nsub* to cover the subexpression that encompasses the whole expression, but that is not parenthesized. It is not believed that this is correct.

The original interpretation request was as follows:

> 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 328–3536), or does it only count explicitly parenthesized subexpressions? This may be the easiest way to rectify the ambiguity.

There is no contradiction. The two paragraphs are discussing two different functions—*regcomp* and *regexec*.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

It is VERY clear that the value for *re_nsub* as set by *regcomp* is the number of actual groupings present in the RE.

In the second paragraph (discussing regexec), it is merely making it clear that *pmatch*[0] describes the entire RE matched, and that *nmatch* must take into account that fact. For example, if there are two parenthesized REs, then one needs to have at least three *regmatch_t*s to have all the submatches recorded.

Assuming for the moment that the entire RE counted as a parenthesized RE, then *re_nsub* would be one higher than today. It is still the case that the entire REs match is recorded in *pmatch*[0], but it would ALSO have to be recorded in *pmatch*[1]! This is because the first subexpression has number 1, and it must be placed in *pmatch*[1].

A new problem has also been noticed, on looking at the POSIX.2 rationale. Note that page 1040, lines 11926 and 11927, suggest that *nmatch* should not be larger that *re_nsub*. This statement seems to be inaccurate, since *nmatch* should equal *re_nsub*+1 if all subexpression data is to be captured. It may, however, have influenced the interpretation.

Interpretation for IEEE Std 1003.2-1992

Interpretation number 2 in IEEE Std 1003.2-1992/INT, March 1994 Edition is incorrect as noted above, and has been withdrawn.

There is an error in the rationale (page 1040, lines 11926–11927), a future revision should change *re_nsub* to *re_nsub*+1.

Rationale for interpretation

This is a defect situation and the previous interpretation has been withdrawn.

It is expected that a future revision of the standard will address the problem in the rationale.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**61**
Topic:	`more -t, vi -t, ex -t`
Relevant Clauses:	5.7, 5.10, 5.18, 5.35
Classification:	1, 4, 5: No change
	2, 3, 6: Ambiguous

Interpretation request

(1) The `more -t` *tagstring* option says that it will write the screenful of the file containing the tag named by the *tagstring* argument.

(2) It is presumed that `more` should behave like `ex` and `vi` as specified in the description of 5.10.7.2.32 of the `ex ta` editing command.

(3) A file named *tags*, which will be searched in the current directory and in other implementation-dependent directories, is used to locate tags for this purpose (and for the `:t` interactive command). The format of that file is expected to be that produced by the `ctags` utility, and if such a file is unreadable for any reason, or the tag cannot be found in the tags file, or tag text cannot be found in the file indicated by the tags file, the behavior is the similar to that of the `vi` utility.

(4) The term *containing the tag* actually means containing the line associated with the tagstring in the tags file.

(5) The `-t` option and the `:t` interactive command not only write the screenful containing the tag, but also remain within the `more(1)` utility for further commands (unless `-e` is used).

(6) Since there could be multiple possible *screenfuls* containing the indicated tag, the screenful written is selected as if the command `/tagtext` were used (that is, centered on the screen where possible unless the line is less than one screenful away from the current position). Basically, the behavior is similar to the behavior of `vi`.

Are these presumptions correct?

Interpretation for IEEE Std 1003.2-1992

(1) This paragraph is a correct statement.

(2) The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(3) The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(4) With the rationale in E.5.18, on page 991, lines 9882–9885, this is a reasonable assumption.

(5) This is a reasonable assumption.

(6) The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number: **62**
Topic: `ex/vi`
Relevant Clauses: 5.10.7.2 and 5.35.7
Classification: 1, 3: Defect
2, 4: No change

Interpretation request

For each of the areas discussed below, it is believed that the intent of POSIX.2 is correct, but that the standard itself is unclear or ambiguous.

(1) *Command Descriptions (5.10.7.2.1 on page 523, lines 1328).* Historically, the beginning/end of the insert was also considered to apply in the test, not just whether or not it was preceded or followed by character that cannot be part of a word. For example, if have the string ABCDEFGH, with the cursor on the C, and type `:ab foo bar<carriage-return>ifoo<esc>`, the abbreviation will be performed.

Suggestion—Reword to make this clear.

(2) *Command Descriptions (5.10.7.2.37 on page 532, lines 1683–1688).* Historically, the ! did not have to immediately precede the file, i.e., `w ! foo` was legal.

Suggestion—Reword to make this clear.

(3) *Command Descriptions (5.10.7.2.44 on page 534, line 1753).* The current breakup of the substitute commands is somewhat confusing. In particular, the substitute command specified on line 1755 is a special case of the main one specified on page 530, line 1597.

Suggestion—Merge the two sections into one section.

(4) *Extended Description (5.35.7 on page 628, lines 4863–4865).* The wording in this paragraph requires that if a `%` or # appears as part of a pathname entered as a command argument that it is expanded to the current or alternate pathname. It is unclear that this wording does not require that a `%` in a substitute pattern be expanded. Is *pathname* a defined term for command arguments? The synopsis lines for things like the `ex` edit command use *file*, not *pathname.*

Suggestion—Reword as necessary. The intent is correct.

Interpretation for IEEE Std 1003.2-1992

(1) The standard states what sequence of characters make up the word, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(2) The standard clearly states in the synopsis a form which corresponds to the example, and conforming implementations shall conform to this.

(3) The standard clearly states a definition for the syntax of substitute and resubstitute, and conforming implementations shall conform to this. Concerns about this issue are being referred to the sponsor.

(4) The standard clearly states the behavior for the occurrence of a `%` or # in a pathname, and conforming implementations shall conform to this.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**63**
Topic:	`ex/vi`
Relevant Clauses:	5.10.7.2.30, 5.35.6.3, 5.35.7.1
Classification:	1, 3–5: Defect
	2: No change

Interpretation request

For each of the areas discussed below, it is believed that, although the standard matches historic practice, historic practice is flawed and that POSIX.2 precludes superior (i.e., reasonable) implementations.

(1) *Command Descriptions (5.10.7.2.30, page 530, line 1596).* The current specification requires that when the substitution is done in visual mode, that the line be redisplayed, with carat characters written underlying the about-to-be-changed string. This is a horrendous user interface, requiring that `vi` not provide the user any context other than the single line, and requiring that the screen be repainted after the changes are made. In addition, it is going to be almost impossible in this model to display changes for search patterns that cross `<newline>` boundaries. The `ex` utility does not even get confirmation of changes to the space between the last character on the line and the end-of-line right now!

Suggestion—Permit implementations to do something reasonable.

(2) *Output Files (5.35.6.3, page 628, lines 4849–4851).* It is reasonable to permit `vi` to add the `<newline>` when the user writes the file, but it is unreasonable to require it. Admittedly, the standard does not describe what happens when `vi` edits a non-text file, but the wording here would require `vi` to decide if a file was a text file and then add a `<newline>` if it was, and not add one if it was not. Otherwise, when vi edits a binary, it has to add an extra character.

Suggestion—Allow the extra character, but do not require it.

(3) `vi` *Command Descriptions (5.35.7.1, page 630, lines 4960–4961).* The current specification requires that the numbered buffers only be available from visual mode. This is not necessary. The reason one cannot get to the numeric buffers from `ex` mode is that the commands like `d2` are ambiguous, since the `d` command takes an optional count. If someone wants to make numeric buffers available from `ex`, that is okay, as long as the current syntax of `ex` is not broken.

Suggestion—Delete this sentence, since it does not add anything to the standard.

(4) *Page backwards (5.35.7.1.1, page 633, line 5079)*
Scroll forwards (5.35.7.1.2, page 633, line 5087)
Scroll forwards by line (5.35.7.1.3, page 633, line 5095)
Page forward (5.35.7.1.4, page 634, line 5104)
Scroll backwards (5.35.7.1.13, page 635, line 5169)
Scroll backwards (5.35.7.1.16, page 636, line 5196)
Move to top of screen (5.35.7.1.58, page 648, line 5611)
Move to bottom of screen (5.35.7.1.62, page 649, line 5646)
Move to middle of screen (5.35.7.1.64, page 649, line 5661)

The historic `vi` had a problem in that all movements were by physical lines, not by logical (i.e., screen) lines. Arguments can be made that this is the right thing to do. For example, single line movements, such as `j` or `k`, should work on physical lines. Commands like `dj`, or `j.`, where "." is a change/delete command, make more sense for physical lines than they do for logical lines.

These arguments, however, do not apply to scrolling commands like `<control-D>` and `<control-F>` when the window is fairly small (a console window, for example), and using physical lines can result in a half-page scroll repainting the entire screen, which is not what the user wanted. Second, if the line is larger than the screen, using physical lines can make it impossible to

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

display parts of the line—there are not any commands that do not display the beginning of the line in historic `vi`, and if both the beginning and end of the line cannot be on the screen at the same time, the user loses. This is even worse in the case of the `H`, `L`, and `M` commands—for large lines, they will all refer to the same line, which results in no movement at all.

Another issue is that page and half-page scrolling commands historically moved to the first non-blank character in the new line. If the line is approximately the same size as the screen, this loses because the cursor before and after a `<control-D>`, will refer to the same location on the screen.

The bottom line is that it is a fundamental flaw (not to mention a security problem) if an editor cannot create lines that it cannot display.

Suggestion—POSIX.2 should permit implementations to do scrolling (`<control-B>`, `<control-D>`, `<control-F>`, `<control-U>`, `<control-Y>`, `<control-E>`), and cursor positioning (`H, L, M`) commands using logical lines, not physical. The only changes required are that POSIX.2 allow this implementation, and note that, in this case, `<control-D>` and `<control-U>` must only set the cursor to the first non-blank character of the line if the current line changed.

`Nvi` is implemented this way, and, so far, no one has ever even noticed.

(5) *Move cursor backwards (5.35.7.1.6, page 634, lines 5118)*
Scroll backwards (5.35.7.13, page 635, line 5169)
Delete word (5.35.7.1.15, page 636, line 5187)

The current standard prevents implementations from allowing users to erase past the current input line, i.e., once the user inserts a new line, they can no longer erase characters before that line. This is wrong, and prevents superior implementations.

Suggestion—Change the specification to permit implementations to erase past the beginning of the line and back to the start of the input text.

Interpretation for IEEE Std 1003.2-1992

(1) The standard states the behavior for the line to be substituted and the current line indicator, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(2) The standard clearly states that a `<newline>` character shall be added, and conforming implementations shall conform to this.

(3) The standard states that numbered buffers 1–9 shall be accessible only from visual mode, and conforming implementations shall conform to this. However, concerns have been raised about this which are being referred to the sponsor.

(4) The standard states the behavior for scrolling, paging, and moving, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(5) The standard states the behavior for moving, scrolling, and deleting, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**64**
Topic:	`ex/vi`
Relevant Clauses:	5.10.7, 5.35.7
Classification:	1, 3, 9–11, 13–15, 19–22 (p3), 23, 24, 27, 30, 33, 35–37, 39–43, 45, 47, 49, 52–57, 59: Defect 2, 12, 28, 34, 44, 46, 48: No change 4–8, 16–18, 22 (p1, p2, p4–6), 25, 26, 29, 50, 51, 58, 60: Unaddressed issue 31, 32, 38: Ambiguous

Interpretation request

For each of the areas discussed below it is believed that the POSIX.2 wording does not match historic practice.

(1) *Extended Description (5.10.7, page 519, lines 1178–1179).* Historically, parsing the command line was a bit more complex than described here.

There are three normal termination cases for an `ex` command. They are the end of the string, or unescaped (by literal next characters) `<newline>` or | characters. Once past the addresses, one can figure out how long the command is. There are three special cases:

(1) The `bang, global, vglobal`, and the filter versions of the read and write commands are delimited by newlines only (they can contain shell pipes).
(2) The `ex, edit,` and visual in `vi` mode commands take `ex` commands as their first arguments.
(3) The `substitute` command takes an RE as its first argument, and wants it to be specially delimited.

Historically, | characters in the first argument of the ex, edit, and substitute commands did not delimit the command. And, in the filter cases for read and write, and the `bang, global, vglobal` commands, they did not delimit the command at all.

For example, the following commands were legal:

```
:edit +25|s/abc/ABC/ file.c
:substitute s/|/PIPE/
:read !spell % | columnate
:global/pattern/p|l
```

It is not quite as simple as it sounds, however. The command:

```
:substitute s/a/b/|s/c/d|set
```

was also legal, i.e., the historic `ex` parser (using the word loosely, since *parser* implies some regularity of format) delimited the REs based on its delimiter and not anything so simple as a command syntax. However, the delimiter had to be nonalphanumeric, so it was possible to detect commands like `sgc`, which were also permitted.

Once these special cases were handled, the rest of the command parsing happened pretty much as described by the standard.

Suggestion—Conform to historic practice. There was no way to escape any character in the `+cmd` argument, as the historic `vi` ignored all escape characters in that area. Fix this by allowing literal `next` characters to escape whitespace characters.

(2) *Addressing (5.10.7.1, page 520, lines 1197 and 1204).* Historically, the RE could be empty, i.e., // was a valid address, and the last RE was used.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Historically, the form `\/` and `\?` were equivalent to `//` and `??`.

Suggestion—Conform to historic practice.

(3) *Addressing (5.10.7.1, page 520, line 1224), Command Descriptions (5.10.7.2, page 521, lines 1241–1252).* Historically, a bunch of the `ex` commands could be given an address of 0, and/or % could be used in an empty file.

Suggestion—Conform to historic practice.

(4) *Command Descriptions (5.10.7.1, page 521, line 1236).* Historical practice is that commands with counts were permitted to go past the end of file. For example, if the file only has 5 lines, the `ex` command `1,6>` fails, but the command `>300` succeeds.

Suggestion—Conform to historic practice.

(5) *Command Descriptions (5.10.7.2, page 521, line 1238).* Historically, whatever the terminal reprint character was, could be entered and it caused `ex` to repaint the current line.

Suggestion—Conform to historic practice.

(6) *Command Descriptions (5.10.7.2, page 521, line 1242).* The wording after range that permits multiple addresses applies to the specification of a line as well as a range. The command `3,5,8=` works historically, using 8 as the command address.

Suggestion—Conform to historic practice.

(7) *Command Descriptions (5.10.7.2, page 521, line 1253).* Historic `ex/vi` permitted commands with counts to go past end-of-file. So, for example, if the file only had 5 lines, the `ex` command `1,6>` would fail, but the command `>300` would succeed.

Suggestion—Conform to historic practice.

(8) *Command Descriptions (5.10.7.2, page 521, lines 1255–1266).* The current standard requires that all + and – offsets follow the flags. Historic practice for dealing with flags in `vi` was bizarre, at best. As an example, the command `:3+++p--#` prints the SIXTH line of the file, with a leading number. This means that the # flag and the + offsets were recognized and the – offsets were not. Regardless, it was not historically required that offsets come after the flags.

Suggestion—As the flags and offsets are unambiguous, document the + and – offsets as flags, and require that they be handled in any order.

(9) *Command Descriptions (5.10.7.2, page 523, line 1309).* Historically, if both a count and range were specified, the offset was from the last address, not the first. The offset in the case was inclusive, therefore `:2,5c7` changes line 5 and six others. There is one exception to this, the join command, for which the offset was not inclusive, therefore the command `2,3join3` joins 3 lines, not 2.

Suggestion—Conform to historic practice.

(10) *Command Descriptions (5.10.7.2, page 523, line 1313).* Historically, if only a line or range was specified while in visual mode, `vi` just moved to the line.

Suggestion—Conform to historic practice.

(11) *Command Descriptions (5.10.7.2, page 523, lines 1316–1317).* Historically, the line number was incremented then the line was displayed.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Suggestion—Conform to historic practice.

(12) *Command Descriptions (5.10.7.2, page 523, line 1319).* Historically, the substitute command was pretty much a special case from beginning to end. For example, `sgc3p` was a legal `ex` command.

Suggestion—Conform to historic practice.

(13) *abbrev (5.10.7.2.1, page 523, line 1328).* Historically, `word` could contain non-word characters, but it was impossible to make the abbreviation fire, i.e., could enter

```
:ab ;ab AB
```

as an abbreviation command, but you could never get it to work because the ; character was not part of a `word`.

Suggestion—It would be possible to permit non-word characters in the word, but that would require some seriously nasty data structures in the implementation to avoid quadratic performance, as one would have to search for a matching string, from the end of the string, on each character. It is also a bit tricky to explain, since one probably does not want to permit this to work for white space characters.

Conforming to historic practice, but requiring an error message if the user tries to enter an abbreviation with an illegal character in word is suggested.

(14) *edit (5.10.7.2.8, page 525, line 1389), Extended Description (5.10.7, page 519, lines 1178–1179).* Historically, the `[+line]` could be any `ex` command, not just an address.

It is not historic practice for the `/pattern` command to start searching from the beginning of the file. The `+cmd` commands were always executed with the current line set to the last line of the file. To make `vi` conform to the current specification, the `+cmd` would have to be run through the `ex` parser. Consider what has to happen when the user enters `+s/foo/bar|/pattern` as the `+cmd` field. It is not going to be pretty.

It is historic practice that additional commands appended to the `edit` command (e.g., `edit file|/pattern` were executed from the first line in the file, and not the last.

To sum up, the `/pattern` search will work if the `wrapscan` option is set or if it is appended to the command instead of being entered as a `+cmd` option.

Suggestion—Conform to historic practice.

(15) *join (5.10.7.2.12, page 526, line 1439).* Historically, counts to `ex` commands were <last_address + count – 1>. To join, it was <last_address + count>.

Suggestion—Conform to historic practice.

(16) *map (5.10.7.2.14, page 527, line 1458).* Historic practice is that keys that map to themselves are returned without further remapping. For example, `:map n nz.` is a legal mapping and will not loop infinitely.

Suggestion—Conform to historic practice.

(17) *move (5.10.7.2.16, page 527, line 1487).* Historically, a target line of 0 placed the lines at the beginning of the file.

Suggestion—Conform to historic practice.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

(18) *next (5.10.7.2.17, page 527, line 1491).* Historically, the `:next` command took an `ex` command preceded by a plus sign, similar to the edit commands.

Suggestion—Conform to historic practice.

(19) *number (5.10.7.2.18, page 527, line 1507).* Historically, the format was `%6d` (that is two spaces). Neither tabs nor a sign were used.

Suggestion—Conform to historic practice.

(20) *Quit (5.10.7.2.23, page 528, line 1539—*`wq`*, 5.10.7.2.37, page 532, line 1673—*`xit`*, 5.10.7.2.38, page 532, line 1539—*`ZZ`*, 5.35.7.1.87, page 654, line 5859).* Historically, `quit`, `wq`, `ZZ`, and `xit` would not quit if there were more files to edit, unless the force option was given or any combination of them were entered twice without any intervening commands.

Suggestion—Conform to historic practice.

(21) *Recover (5.10.7.2.25, page 529, line 1557).* The `recover` command historically took a force flag that behaved as other `vi` force flags. The wording about "if the current buffer has been modified ..." should be duplicated for `:recover`. The `autowrite` command did not affect `:recover`.

Suggestion—Conform to historic practice.

(22) *substitute (5.10.7.2.30, page 530, line 1596).* The delimiter of a substitute command could be any nonalphanumeric character.

Historic `ex` accepted any of the following forms:

`:s/abc/def/`	change `abc` to `def`
`:s/abc/def`	change `abc` to `def`
`:s/abc/`	delete `abc`
`:s/abc`	delete `abc`

If the pattern string was empty, the last RE specified to any command was used, not just the last substitution RE.

Historically, substitute commands set the search direction.

In historic System V, if the entire replacement pattern was %, the last replacement pattern was used.

There is no explanation of quoting in the substitute command. Generally, escapes in the command were only interesting if they escaped special characters. This means that `s/A/\\\\f` replaced `A` with `\\f`.

Suggestion—Conform to historic practice.

(23) *visual (5.10.7.2.36, page 531, line 1661).* Historically, `:visual` was different if executed while in `visual` mode than if executed while in `ex` mode.

In `visual` mode, it was equivalent to `edit[!] [+cmd] [file]`.

Suggestion—Conform to historic practice.

(24) *Escape (5.10.7.2.41, page 533, line 1724).* Historically, the program named by the shell option was run with the `-c` *flag* followed by the rest of the arguments as a single `argv` entry. The **SHELL** environment variable does not enter the picture other than as the initial value of the `shell` option.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Historically, the line was set to the first line of the range if a range was specified, e.g., `3,5!date` while on line 10 changes the current line to 3.

Suggestion—Conform to historic practice.

(25) *Shift left (5.10.7.2.42, page 534, line 1742), Shift right (5.10.7.2.43, page 534, line 1748).* Historically, the < and > characters could be repeated to shift left or right a specific number of indentation levels.

Suggestion—Conform to historic practice.

(26) *Shift left (5.10.7.2.42, page 534, line 1742), Shift right (5.10.7.2.43, page 534, line 1748).* The shift command in historic `ex` had the usual bizarre collection of cursor semantics. If called from vi, the cursor was repositioned to the first non-blank character of the lowest numbered line shifted. If called from ex, the cursor was repositioned to the first non-blank of the highest numbered line shifted.

Suggestion—If the cursor is not part of the set of lines that are moved, move it to the first non-blank of the last line shifted. (This makes `:3>>` in `vi` work reasonably.) If the cursor is part of the shifted lines, it does not get moved at all. This permits shifting of marked areas, i.e., `>'a.` shifts the marked area twice, something that could not be done with historic vi.

(27) *Edit Options (5.10.7.5, page 536, line 1817).* Historically, whitespace characters in set commands were protected by backslashes.

Suggestion—Conform to historic practice.

(28) *list (5.10.7.5.7, page 537 1886).* Historically, the list option only affected tab characters and the end-of-line display. More specifically, when entering an `ex` command in visual mode, it only affected tab characters.

Suggestion—Conform to historic practice.

(29) *report (5.10.7.5.16, page 539 1936).* Historically, yanks (which do not modify lines) were also reported by the editor. In addition, the test for yanked lines was >=, not > as it was for everything else.

Suggestion—Conform to historic practice.

(30) *wrapmargin (5.10.7.5.30, page 541, line 2012).* Historically, if the user entered a `<blank>` that caused the lines to split, and then entered a `<space>`, it was discarded.

Suggestion—Conform to historic practice.

(31) *Asynchronous Events (5.35.5.4, page 627, lines 4835–4837).* Historic practice was that the buffer was saved unless a complete write had been accomplished, i.e., the whole file had to have been written, not just part of it. Why is there any reference here to the `e` command.

This problem is repeated throughout the standard, basically, any `ex` or `vi` command (e.g., `:edit`, `ZZ`, etc.) that discusses whether or not a file is written if it has been modified, needs to be changed.

Suggestion—Change all of these to say something like "if the file has been modified since it was last completely written...," and add something that defines a "complete write."

(32) *`vi` Command Descriptions (5.35.7.1, page 629, line 4904).* The current column is documented badly. The line is documented as a physical line, i.e., there is no attempt to describe that a "current line" may change because of an `l` command on a line that wraps. The column, however, is documented as a logical column, i.e., the actual column on the screen that the cursor is on instead of

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

the character it is on. The confusion comes as a result of it not being documented that the column is longer than a screen line.

For example, in the `<control-H>` command (see page 634, line 5117) note that the "Synopsis paragraph" says that the cursor moves "back count characters," but the "Current column" paragraph says that the column is set to "column—the number of columns occupied by count characters."

Note that the specification currently is correct in this regard, this change does not have to be made. But, it would be significantly simpler and clearer to describe the column in terms of the character that the cursor sits on, not the column number.

Suggestion—Reword the command descriptions that describe the column in terms of the character that the cursor is on.

What may not be documented correctly is that if the line changes, for example the `j` command, the current column may change too, and the standard does not acknowledge this. (Line 5135, on page 634, the `j` command states that the current column is unchanged.) The problem is if one has the file:

```
ABCDEFGH
foo^Abar
```

with the cursor on the `D`, and one enter the `j` command. Because of the display rules, the cursor must move to either the `o` in `foo` or the `A` in `^A`. The specification currently does not handle this.

In `nvi`, there is the concept (in the documentation) of a most attractive cursor position. What happens is that when the cursor is forced to change lines, it tries to get as close as possible to the previous logical column, which is a function of the screen column and line wrap. This seems to work well.

(33) *`vi` Command Descriptions (5.35.7.1, page 630, lines 4956–4964).* The current `vi` specification is a bit confusing on the relationship between the numeric, unnamed and user specified buffers. The reason for this confusion was that historic `vi` tossed the contents of the unnamed buffer after just about every operation. For example, if enter `add|p`, the delete can be recovered, but in `add|"ap|p`, the put of the unnamed buffer will fail. However, more to the point, note that the commands `add|p`, `add|"ap`, and `add|"1p` all recover the deleted line.

In addition, the current `vi` specification requires that any and all deleted text be placed into the numeric buffers. Historic practice was that deleted text was only placed in numeric buffers if at least one line was deleted from the file or the change/delete crossed a line boundary.

Suggestion—The way to document this is that if the user specifies a buffer, the text is copied into it. If the delete is a line or if it crosses a line boundary, it is placed into both the unnamed buffer and the numeric buffers. Do not document that historic `vi` tossed the unnamed buffer more often than it should. It is a bug, not a feature.

(34) *`vi` Command Descriptions (5.35.7.1, page 630, lines 4960–4971).* The current `vi` specification disallows users from entering text into the numeric buffers. Historic practice was that this was permitted, although, to be fair, `vi` could then be convinced to do genuinely strange things. (It behaved differently if you deleted text than if you yanked text; in the latter case, the text was appended to the buffer instead of replacing the contents.)

Suggestion—Specify that numeric buffers be treated no differently than any other buffer when specified by users. Document that the buffer rotation happen before the replacement of, or the appending to, the contents of the buffer.

(35) *`vi` Command Descriptions (5.35.7.1, page 631, lines 4989–4999).* The current `vi` specification requires that motion regions be "from the current cursor position to just before the cursor position indicated by the motion command." This does not work for backward motions. Historic practice was that the region was from the earliest cursor position in the file, (regardless of whether that was the

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

starting cursor position or the one indicated by the motion command) to just before the later cursor position. Naturally, there were some special cases where this was not true, but that is the general rule. As an example, note the commands `fgdTc` on the string `abcdefg` or the commands `fgd?c` on the same string.

In addition, historic practice had several special cases for motion commands. For example, `dl` succeeds when the cursor is on the last character of the line, and, obviously `l` fails.

Suggestion—Conform to historic practice.

(36) *`vi` Command Descriptions (5.35.7.1, page 631, lines 4995–4998).* The determination of whether or not a command affected `lines` was actually a lot trickier than is described here. (It is bad enough that `nvi` does it on a command-by-command basis—there was an inability to construct general rules that were close enough to historic practice.)

As a single example, consider searches as a motion component. The standard notes (page 642, line 5402) that a delta to the search makes it line oriented. However, it does not note that:

(1) + searches that start and end at the first column of a line become line mode operations.

(2) + backward searches starting at column 0, and forward searches ending at column 0 are corrected to the last column of the previous line.

Note also that forward and backward searches can occur (based on the `wrapscan` option) for any search command.

Suggestion—Document on a command-by-command basis what can cause motions to become line oriented.

(37) *`vi` Command Descriptions (5.35.7.1, page 631, lines 5000–5012).* Were `<control-B>`, `<control-D>`, `<control-E>`, `<control-F>`, `<control-T>`, `<control-U>`, or `<control-Y>` were historical cursor motion commands?

It is believed that `+`, `_`, and `h` were historical cursor motion commands.

Suggestion—Delete the first set, add the second set.

(38) `vi` Command Descriptions (5.35.7.1, page 631, lines 5013–5015. This paragraph does not fully describe historic practice. In commands that take motion arguments, any count provided was applied to the motion command, e.g., `2cw` was identical to `c2w`. If counts were provided for both commands, the effect was multiplicative, e.g.,. `4cw` is identical to `2c2w`.

Suggestion—Conform to historic practice.

(39) *`vi` Command Descriptions (5.35.7.1, page 632, lines 5024–5025).* Were `<form-feed>` characters traditional delimiters (4BSD and SunOS 4.1 were tested)?

Suggestion—No objection if it is useful, but is it widespread historic practice?

(40) *`vi` Command Descriptions (5.35.7.1, page 632, line 5036).* Allowing any number of double quotes is actually not historic practice, although it is historical documentation. Allowing any number of single quotes is both historic practice and documentation.

Suggestion—Add single quotes to the list. Note in the rationale that POSIX.2 is requiring that the documentation be supported.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

(41) `vi` *Command Descriptions (5.35.7.1, page 632, lines 5041–5048).* This paragraph does not fully describe historic practice. Words were delimited by the beginning of the file as well as the end. (The bigword definition gets it right, incidentally.)

Suggestion—Conform to historic practice.

(42) `vi` *Command Descriptions (5.35.7.1, page 632, lines 5053–5056).* Scrolling if it is (window – 1) lines away is not wanted, only if it is half a screen or less. Historic `vi` would repaint at far less provocation than this states. In 4BSD's `vi`, a 9 line movement at the bottom of a 32 line screen was sufficient, and in SunOS 4.1, a 12 line movement in the bottom of a 31 line screen was sufficient. This is also terminal/hardware specific. It can be faster to clear and repaint the screen than to scroll the terminal, particularly over slow lines.

Suggestion—Reword this to leave it up to the implementation, but suggest that window / 2 be scrolled.

(43) `vi` *Command Descriptions (5.35.7.1, page 632, lines 5059–5064).* Historic practice as to whether or not a motion command failed at the top or bottom of the screen was very command specific. There are not any general rules that you can specify. For example, `<control-U>` fails if the cursor is on line 1 of the file, but succeeds if the cursor is on line 2. For most movements, the end/beginning of the file was a movement sink, but it was an error if the cursor was already there.

An even worse example is the `]]` command fails if it is on the last line of the file. The `}` command succeeds on the last line and moves to the last character of the last line, only failing if it is already on the last character of the last line. This behavior is mirrored on the first line of the file with the reverse directions of these commands.

Suggestion—Specify the motion failure modes on a command-by-command basis.

(44) *Move cursor backwards (5.35.7.1.6, page 634, line 5118), Scroll backwards (5.35.7.13, page 635, line 5169), Delete word (5.35.7.1.15, page 636, line 5187).* Historically, the erase commands could not erase autoindent characters (or characters inserted using the `<control-T>` command). This is a good change, and should be permitted.

Suggestion—Note in the Rationale that the erase commands are now permitted to erase any character, no matter how it was entered. Change the specification to permit implementations to not erase any auto-indent characters or characters inserted using `<control-T>`.

(45) *Replace text with results from shell command (5.35.7.1.9, page 637, line 5238).* Historic practice is to use the contents of the shell option, not the **SHELL** environment variable.

Suggestion—Conform to historic practice.

(46) *Move forwards with tabs (5.35.7.1.12, page 635, lines 5164–5166).* Historically, `<control-T>` only worked if nothing other than auto-indent characters were in the line, similar to `<control-D>` erasure. The change is correct, and should be retained.

Suggestion—Note the change in the Rationale.

(47) *Replace text with results from shell command (5.35.7.1.9, page 637, line 5238).* Historic practice is to use the contents of the shell option, not the **SHELL** environment variable.

Suggestion—Conform to historic practice.

(48) *Move forwards with tabs (5.35.7.1.12, page 635, lines 5164–5166).* Historically, `<control-T>` only worked if nothing other than auto-indent characters were in the line, similar to `<control-D>` erasure. The change is correct, and should be retained.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Suggestion—Note the change in the rationale, and that it was deliberate.

(49) *Move forwards with tabs (5.35.7.1.12, page 635, lines 5164–5166).* Historically, `<control-T>` did not move forward shiftwidth characters, but moved to the next shiftwidth column boundary.

Suggestion—Conform to historic practice.

(50) *Scroll backwards (5.35.7.1.13, page 635, line 5169).* Historically, `<control-U>` in text input mode erased the current line.

Suggestion—Conform to historic practice.

(51) Return to previous context (at beginning of line) (5.35.7.1.23, page 639, line 5292), Return to previous context (5.35.7.1.24, page 639, line 5306). Historic practice was that the commands `and` worked, too.

Suggestion—Conform to historic practice.

(52) *Move back to beginning of sentence (5.35.7.1.28, page 640, line 5345), Move forwards to beginning of sentence (5.35.7.1.29, page 640, line 5351), Move back to preceding paragraph, (5.35.7.1.30, page 640, lines 5357), Move forwards to next paragraph, 5.35.7.1.31, page 641, line 5365).* There is no discussion of how empty lines work with `(`, `)`, `{`, and `}`.

Suggestion—Match historic practice, documenting that groups of empty lines count as a single sentences and paragraphs.

(53) *Move backwards to preceding word (5.35.7.1.47, page 645, line 5509), Move backwards to preceding bigword (5.35.7.1.47, page 645, line 5521), Move to end-of-word (5.35.7.1.53, page 646, line 5564), Move to end-of-bigword (5.35.7.1.54, page 647, line 5576), Move to beginning of word (5.35.7.1.80, page 653, line 5796), Move to beginning of bigword (5.35.7.1.81, page 653, line 5807).* The current specification says that for all word commands, an empty/blank line is considered to contain exactly one word.

This is not historic practice, although historic practice was far from consistent. In historic vi, the `w`, `W`, and `B` commands would treat groups of empty lines as individual words, i.e., the command would move the cursor to each new empty line. The `e` and `E` commands would treat groups of empty lines as a single word, i.e., the first use would move past the group of lines. The `b` command would just beep, or, if done from the start of the line as part of a motion command, would cause bizarre behavior. If the lines contained only whitespace characters, the `w` and `W` commands would just beep , and the `B`, `b`, `E`, and `e` commands would treat the group as a single word, and the `B` and `b` commands would treat the lines as individual words.

Suggestion—Since some of the commands already treat groups of empty lines and whitespace only lines as a single word, and the `(`, `)`, `{`, and `}` commands behave that way, too, and it is probably more useful to users that groups of empty lines be skipped, change the standard to require that empty lines, or lines with only whitespace characters are treated as a single word. Note the change in the rationale. (This is the way that `nvi` behaves, incidentally, with no complaints so far.)

(54) *`vi` Command Descriptions (5.35.7.1, page 633, lines 5069–5078).* These rules are not historically correct. For example, `dl` is permitted at the end of a line. A guess is that the only thing that can be done is to describe the boundary cases on a command-by-command basis. (That is how it had to be implemented.)

Suggestion—Conform to historic practice.

(55) *`vi` Command Descriptions (5.35.7.1.1, page 633, line 5079, 5.35.7.1.4, page 634, line 5104).* The `^B` and `^F` calculations are not historically correct.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

The correct `^F` calculation is:

```
top_line = top_line + count * text_rows -2;
```

The correct `^B` calculation is:

```
top_line = (top_line - count * text_rows) + 2;
```

A simpler approach is to note that both `^B` and `^F` scroll:

```
count * text_rows -2
```

lines.

Suggestion—Conform to historic practice.

(56) `vi` *Command Descriptions (5.35.7.1.27, page 640, line 5340).* Historically, the ^ command took a count that it then ignored.

Suggestion—Conform to historic practice.

(57) `vi` *Command Descriptions (5.35.7.1.32, page 641, line 5378).* Historically, the | command was placed on the character that spanned the column, not the one following it.

Suggestion—Conform to historic practice.

(58) `vi` *Command Descriptions (5.35.7.1.36, page 642, line 5403, 5.35.7.1.42, page 644, line 5466).* Historically, both / and ? took counts.

Suggestion—Conform to historic practice.

(59) `vi` *Command Descriptions (5.35.7.1.43, page 644, line 5470).* Historic practice was that `<newlines>` were (logically) added to each line in a line oriented buffer, and to all but the last line of a character-oriented buffer. For example, if buffer a contained the following:

```
:3p|4p
:5p
```

cut with `ay6w`, the `3p` and `4p` are executed, and the cursor waits after the `5p` has been displayed on the colon command line.

Suggestion—Conform to historic practice.

(60) `vi` *Command Descriptions (5.35.7.1.86, page 654, line 5848).* The `z` command historically had + and ^ arguments.

The + character was the same as `<newline>` if `count1` was specified, otherwise, it displayed the next screen after the current one, like `^F`, with the exception that there was no overlap between the screens.

The ^ character displayed the screen after the screen with `count1` on the bottom line, if `count1` was specified, otherwise, it displayed the screen after the current one, like `^B`, with the exception that there was no overlap between the screens.

Suggestion—Conform to historic practice, or, note that those characters should not be used for other extensions.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation for IEEE Std 1003.2-1992

(1) The standard states rules and behavior for the `ex` parser, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(2) The standard clearly states that a null BRE (//) shall be equivalent to the last BRE encountered, and conforming implementations shall conform to this.

(3) The standard states the behavior for addressing in `ex`, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(4) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(5) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(6) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(7) The request is substantially identical to interpretation number 4 (above), and the resolution of that interpretation applies in this case.

(8) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(9) The standard states the behavior for ranges and counts in `ex`, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(10) The standard states the implied behavior when only a line or range is specified in `ex`, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(11) The standard states the sequence of behavior when no range or count is specified and the command line is a blank line, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(12) The standard clearly states the syntax when a object follows the command name, and conforming implementations shall conform to this.

(13) The standard clearly states the behavior for word in the abbrev command, and conforming implementations shall conform to this. Concerns about this issue are being referred to the sponsor.

(14) The standard states the behavior for the `[+line]`, `/pattern`, `?pattern`, and appended command execution, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(15) The standard states the behavior for counts in the `ex` command, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(16) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(17) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

(18) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(19) The standard states the format for number, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(20) The standard states the behavior for quit in `ex`, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(21) The standard states the syntax for the `recover` command, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(22) (Paragraph 1) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(Paragraph 2) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(Paragraph 3) The standard states the pattern to use if the pattern string is empty, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(Paragraph 4) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(Paragraph 5) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(Paragraph 6) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(23) The standard states the behavior for the visual command, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(24) (Paragraph 1) The standard states the relationship between escape and **SHELL**, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(Paragraph 2) The standard states the behavior for `escape` and `ranges`, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(25) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(26) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(27) The standard clearly states on page 529, line 1582, that any `<blank>`s in strings can be included as is by preceding each such character with a backslash, and conforming implementations shall conform to this.

(28) The standard clearly states the behavior for list, and conforming implementations shall conform to this.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

(29) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(30) The standard states the effect defined for `wrapmargin`, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(31) The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(32) The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(33) The standard states the behavior of numeric buffers, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(34) The standard clearly states that "text cannot be placed directly into the numbered buffers," and conforming implementations shall conform to this.

(35) The standard states the behavior of the motion argument, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(36) The standard states the rules for determining the range of `lines` that will be affected by commands, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(37) The standard states that `<control-B>`, `<control-D>`, `<control-E>`, `<control-F>`, `<control-T>`, `<control-U>`, `<control-Y>`, `+`, and `_` are cursor motion commands, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(38) The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(39) The standard states that a `<form-feed>` character is a delimiter, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(40) The standard states that any number of double-quote characters are allowed, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(41) The standard states the definition for `word`, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(42) The standard states rules for scrolling within `vi`, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(43) The standard states the rules for motion, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(44) The standard clearly states the behavior for delete word, and conforming implementations shall conform to this.

(45) The standard states that the **SHELL** environment variable is to be used, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

(46) The standard clearly states the behavior for `<control-T>`, and conforming implementations shall conform to this.

(47) The request is substantially identical to interpretation number 45 (above), and the resolution of that interpretation applies in this case.

(48) The request is substantially identical to interpretation number 46 (above), and the resolution of that interpretation applies in this case.

(49) The standard states that the cursor is to be moved forwards shiftwidth positions, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(50) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(51) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(52) The standard states the behavior for areas of text in the paragraph and sentence descriptions on page 632, lines 5019–5023 and 5024–5029, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(53) The standard states that an empty or blank line shall be considered to contain exactly one word, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(54) The standard states the rules for handling position exceptions, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(55) The standard states the calculations for `<control-B>` and `<control-F>`, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(56) The standard clearly states the definition for the ^ command, and conforming implementations shall conform to this. Concerns about this issue are being referred to the sponsor.

(57) The standard states that the cursor shall be placed on the character following the *count* th column position, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(58) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

(59) The standard states the behavior for `<newlines>` and buffers, and conforming implementations shall conform to this. However, concerns have been raised about this, which are being referred to the sponsor.

(60) The standard does not speak to this issue, and as such no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**65**
Topic:	ex—Command Descriptions
Relevant Clauses:	5.10.7.2
Classification:	Defect

Interpretation request

The current standard incorrectly documents which address is used for the starting line, when both a line range and count are specified. Historical practice has it as the last address in a range specification, not the first.

To remedy this problem, could the word *first* on page 523, line 1311, be replaced with *last*?

Interpretation for IEEE Std 1003.2-1992

The request is substantially identical to interpretation number 64 question 9 in this document, and the resolution of that interpretation applies in this case.

Rationale for interpretation

None.

IEEE

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**66**
Topic:	`more`—Extended Description
Relevant Clauses:	5.18.7
Classification:	No change

Interpretation request

The standard appears unclear as to the effect of a multibyte character followed by a `<backspace>` character; if the `<backspace>` is written directly to the terminal, then the terminal driver/hardware could treat the `<backspace>` in one of two ways.

(1) Move the cursor left one screen column.
(2) Move the cursor left one character (so that subsequent characters overwrite).

The standard appears to expect case 2, though the use of the word "generally" on page 568, line 2904, does imply some exceptions and therefore permit case 1. What handles moving the cursor one character left—`more` or the terminal? If the answer is the terminal then should POSIX.2 be corrected since behavior outside their control is being mandated?

Would it be possible to replace lines 2903–2905 on page 568 with:

> Other `<backspace>` sequences are written to the terminal so that for each <backspace>, the terminal's cursor is moved left the width of the character preceding the terminal's cursor until the left margin is reached. The effect of a <backspace> when the cursor is at the left margin is undefined.

Interpretation for IEEE Std 1003.2-1992

The definition of `<backspace>` is on page 16 lines 266–267. Although there is no rationale for `<backspace>` in `more`, the rational for backspace in `fold` applies here as well. The rationale for `fold` is in E.4.25 on page 908, lines 6357–6368.

In terms of what handles moving the cursor one character left, neither one. It is the application producing the text file that `more` is asked to display.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**67**
Topic:	`mkdir -m mode`
Relevant Clauses:	4.41
Classification:	Ambiguous

Interpretation request

This problem was encountered while using the POSIX.2 Commands test suite.

Should `mkdir -m mode` utilize the current umask in setting permissions? It is proposed that it does not.

The test assertion in IEEE P2003.2 Draft 9, February 1994, say it should be based on the DESCRIPTION for `mkdir`, which says it "shall perform actions equivalent to the POSIX.1 *mkdir*()."

However, existing practice in System V (on which the rationale says this option was added) is to set the permission to the explicit value specified by the mode argument to -m; and the text in lines 6950–6956 on page 358 state that "The *mode* option-argument shall be the same as the *mode* operand defined for the `chmod` utility." The `chmod` utility ignores the umask.

So for an example, say

```
$ umask 22
$ mkdir -m 456 foodir
$ ls -ld
```

gives (on System V)

```
dr--r-xrw-   2 andrew  relx      512 Jul 1 04:29 foodir
```

whereas IEEE P2003.2 Draft 9, February 1994, wants

```
dr--r-xr--   2 andrew  relx      512 Jul 1 04:29 foodir
```

As a user, the System V behavior seems more intuitive. Typical usage has been to set the mode explicitly in a single command using the -m option, rather than call `umask` and then calling `mkdir -m xxx`.

Understanding of the rationale in E.4.41 on page 927, lines 7184–7187, "For example, by default, the mode of the directory is affected by the file mode creation mask" is that this does not apply to the -m case, which is not the default.

Interpretation for IEEE Std 1003.2-1992

The standard is unclear on this issue, and no conformance distinction can be made between alternative implementations based on this. This is being referred to the sponsor.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Interpretation Number:	**68**
Topic:	cp pathname arguments
Relevant Clauses:	2.13.1
Classification	No change

Interpretation request

POSIX.2 defines pathname resolution by reference to POSIX.1, and specifically refers to the definition of PATH_MAX in POSIX.1 (see subclauses 2.2.2.121, 2.2.2.123 and 2.13.1 of POSIX.2). Those utilities specified in POSIX.2 that take pathname arguments are required to be able to resolve pathnames that (including an implied trailing null byte) are no more than PATH_MAX bytes in length, assuming no other system resource limits arise.

The issue to be resolved has to do with files that are implicitly referenced by utilities with pathnames that are longer than PATH_MAX. Specifically, consider the command line

cp *filename dirname*

where *dirname* is a pathname of length PATH_MAX-1 that refers to a directory, and *filename* names a regular file. The semantics of the cp utility require that (assuming no access or permission issues interfere) cp create a copy of *filename* in *dirname.* No pathname argument exceeds PATH_MAX-1 bytes; however, this command implies a pathname of the target file, *dirname/filename*, that exceeds PATH_MAX bytes. The reference is somewhat more than implicit, since the description of cp (see POSIX.2, subclause 4.13.2, pages 231–232, lines 2589–2591) states

> The cp utility shall copy the contents of each *source_file* to the destination path named by the concatenation of *target*, a slash character and the last component of *source_file.*

Is cp required to work in this case?

Note that a similar question can be asked with respect to the mv and ln utilities and perhaps others.

Interpretation for IEEE Std 1003.2-1992

The wording in cp as specified in this interpretation request clearly allows an implementation of cp to call open with the pathname created by concatenating *dirname*, slash ("/"), *filename* and in cases where this string is longer than PATH_MAX bytes there is no requirement that this should succeed.

Rationale for interpretation

None.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.

To order IEEE standards by phone, call 1-800-678-IEEE (US), 8 a.m.–4:30 p.m. (EST).
Outside of the US and Canada, call 1-908-981-1393.
To order by fax, dial 1-908-981-9667.

ISBN 1-55937-476-4

90000

9 781559 374767

ISBN 1-55937-476-4

Authorized licensed use limited to: Ryerson University Library. Downloaded on November 06,2016 at 01:32:40 UTC from IEEE Xplore. Restrictions apply.