IEEE P1003.2a Draft 8 – December 1991

STANDARDS PROJECT

Draft Standard for Information Technology — Portable Operating System Interface (POSIX) Part 2: Shell and Utilities — Amendment 1: User Portability Extension (UPE)

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

Work Item Number: JTC 1.22.21.2

Abstract: ISO/IEC 9945-2: 199x (corresponding to the merger of IEEE Std 1003.2-199x and IEEE Std 1003.2a-199x) is part of the POSIX series of standards for applications and user interfaces to open systems. It defines the applications interface to a shell command language and a set of utility programs for complex data manipulation. When the User Portability Utilities Option is included, the standard also defines a common environment for general purpose time-sharing users on character-oriented display terminals.

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

P1003.2a / D8
December 1991

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

Editor's Notes

This section will not appear in the final document. It is used for editorial comments concerning this draft.

**The IEEE ballot for Draft 8 is due at the IEEE Standards Office on 8
24 January 1992. Please read the balloting instructions in Annex F. 8**

This is the third IEEE recirculation draft of the UPE. 8

This document is also registered as ISO/IEC CD Amendment 1 to CD 9945-2. The 8
international balloting period is unrelated to the IEEE balloting. Member bodies, 8
please consult any accompanying materials from SC22. Also, please read the 8
remainder of these Editor Notes to see explanations of stylistic differences 8
between a draft and the final standard (copyright notices, inline rationale, etc.). 8

The IEEE balloting will be on hiatus during the international balloting period, 8
which is probably scheduled to complete at the May 1992 WG15 meeting. This is 8
in accordance with the WG15 Synchronization Plan, which calls for coordinated 8
balloting to result in the approval of an IEEE/ANSI standard that is identical to 8
the ISO/IEC Draft International Standard (DIS) or Draft Amendment (DAM). 8
There will be a final recirculation of a full draft (9) to the IEEE balloting group 8
before it is sent to the Standards Board. 8

Draft 8 contains diff marks (''8'')in the right margin denoting changes from Draft 8
7 to Draft 8; 7 marks denote changes from Draft 6 to Draft 7. All diff marks from 8
previous drafts have been removed to focus balloting group attention on the 8
remaining issues to be considered in the recirculation. Editorial changes such as 7
typos, grammatical errors or changes, changes in cross references, and removal of editorial notes are not diff-marked. Please note that it is not always feasible to get the diff marks exactly right; they will sometimes start or end a line too soon.

We have been directed that all sentences that are currently declarative must be changed to use the ''shall'' form if they pose a requirement: ''The status is zero'' → ''The status shall be zero.'' It is a tedious process to do all these translations and they are not complete. They will completed on a draft-by-draft basis. In the meantime, please assume that all declarative sentences mean to use ''shall'' and treat them as either implementation or application requirements unless they specifically say ''may,'' ''should,'' or ''can.''

This draft makes assumptions about the organization and content of POSIX.2 8
Draft 11.2 (also known as ISO/IEC CD 9945-2.2, JTC 1 SC22 N1063), which is being 8
balloted by a slightly different group of people. If you are not in the POSIX.2 bal- 8
loting group (i.e., if you did not receive Draft 11.2), you can purchase the draft by 8
contacting:

IEEE Publications 8
P.O. Box 1331
445 Hoes Lane
Piscataway, NJ 08855-1331

1 (800) 678-IEEE 8
+1 (908) 562-3800 (outside US) 8

(Or see the online access instructions below.) 8

Since portions of this standard are meant to be modifications of the base POSIX.2 standard, the draft headings in Sections 1 through 4 have been set up to match the affected clauses and still go into the table of contents. There are therefore gaps in the clause numbers.

Please report typographical errors to:

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

(Electronic mail is preferred.)

The copying and distribution of IEEE balloting drafts is accomplished by the Standards Office. To report problems with reproduction of your copy, contact: 7

Anna Kaczmarek 7
IEEE Standards Office
P.O. Box 1331
445 Hoes Lane
Piscataway, NJ 08855-1331

+1 (908) 562-3811 7
FAX: +1 (908) 562-1571

This draft and the current P1003.2 draft are available in various electronic forms to assist the review process. Our thanks to Andrew Hume of AT&T Bell Laboratories for providing online access facilities. Note that this is a limited experiment in providing online access; future ballots may provide other forms, such as diskettes or a bulletin board arrangement, but the instructions shown here are the only methods currently available. Please also observe the additional copyright restrictions that are described in the online files.

Assuming you have access to the Internet, the scenario is approximately

```
ftp research.att.com # research's IP address is 192.20.225.2
<login as netlib; password is your email address>
cd posix/p1003.2a/d8
get toc index
binary
get p11-20.Z
```

The draft is available in several forms. The table of contents can be found in `toc`, pages containing a particular section are stored under the section number, sets of pages are stored in files with names of the form `p`*n*−*m*, and the entire draft is stored in `all`. By default, files are ASCII. A `.ps` suffix indicates PostScript. A `.Z` suffix indicates a `compress`'ed file. The file `index` contains a general description of the files available.

These files are also available via electronic mail by sending a message like

```
send 4.55 5.6 from posix/p1003.2a/d8
```

to `netlib@research.att.com`. If you use email, you should *not* ask for the compressed version. For a more complete introduction to this form of *netlib*, send the message

```
send help
```

POSIX.2a Change History

This section is provided to track major changes between drafts. Since it was first added in Draft 6, earlier entries omit some degree of detail.

Draft 8 [November 1991] Fourth IEEE ballot (third recirculation). First 8
circulation as an ISO/IEC Proposed Draft Amendment: CD 8
Amendment 1 to CD 9945-2. 8

- `sh` (4.55): `emacs` command-line editing mode removed. 8
- `at` (5.2): Numerous clarifications to the *timespec* operands. 8
- `ex` (5.10): Addition of tilde resubstitute command; 8
  clarification of startup sequence and `-s` option; explanation of 8
  `control-Q` effects (changes also affect `vi`). 8
- `fc` (5.12): Clarification of command history numbering and 8
  editing multiple commands. 8
- `mesg` (5.17): Modified algorithm for terminal selection. 8
- `more` (5.18): Changed `-p` *pattern* to `-p` *command*; disal- 8
  lowed data modification when used as a filter; moved standard 8
  output description to standard error. 8
- `patch` (5.22): Removed `-p`, `Prereq:` feature. 8
- `time` (5.29): Modified to allow the KornShell practice that 8
  times a pipeline. 8
- Miscellaneous clarifications and corrections to various utili- 8
  ties. 8

Draft 7 [July 1991] Third IEEE ballot (second recirculation). 7

- Changes to text in Sections 2 through 5 that clarify which 7
  features are being added to support the UPUO (and are thus 7
  optional for POSIX.2-only systems). 7
- General cleanup of the defaults values and usage of **TERM**, 7
  COLUMNS, and **LINES**. 7
- Clarifications to Section 2 and the `nice` and `renice` utilities 7
  on the nature of ‘‘system scheduling priority’’ (versus other 7
  scheduling policies on the system) and how privileges are 7
  involved. 7
- Clarifications to various uses of the term ‘‘nonprintable’’; the 7
  POSIX.2 term ‘‘printable’’ does not include any white-space 7
  characters other than `<space>`, causing some synchronization 7
  problems. 7
- Modifications to the `uuencode` algorithm to address differ- 7
  ences between ISO/IEC 646 {1} and other, incompatible, code 7
  sets. 7

- Modifications to the descriptions of the `mailx` mailbox for- 7
  mats. 7

- Numerous clarifications and minor modifications to `ex`, `more`, 7
  and `vi`, mostly concerning screen movements. 7

- Miscellaneous clarifications and minor modifications to other 7
  utilities. 7

Draft 6 [March 1991] Second IEEE ballot (first recirculation).

- The `compress`, `lint89`, `passwd`, `uncompress`, and `zcat` utilities were removed due to balloting opposition. See the rationale in clause E.4.

- The {POSIX 2_CHAR _TERM } option was added.

- An asynchronous job notification option has been added to the `set` built-in. This is done through both `set -o notify` and `set -b`.

- Global shell variables **RANDOM** , **SECONDS** , and _ have been removed.

- The `nice` and `renice` utilities have changed from the use of ‘‘nice value’’ to ‘‘system scheduling priority.’’ They have also removed the {MAX_NICE }, and {MIN _NICE } and related limits. The `renice` utility has major interface changes to conform to the utility syntax guidelines

- The `mailx` and `patch` utilities have had major clarifications applied.

- The tags features of `ex`, `more`, and `vi` have been marked as belonging to the Software Development Option.

- The `df` and `du` were changed to use consistent, historical 512-byte reporting units and given a `−k` option to report in kilobytes.

- The `file` utility has extra defined strings and an explanation of `/etc/magic` in the rationale.

- The `man` utility added keyword searching and paging abilities.

- The `nm` and `ps` utilities have had major interface changes and clarifications.

- The `uuencode` and `uudecode` utilities have had clarifications added in the processing of octets versus bytes.

- The `unalias` utility now has the `a` option added for removing all aliases.

- Miscellaneous clarifications and minor technical changes have been made to various utilities.

— Instances of the verb ‘‘print’’ were changed to ‘‘write’’ as part of a general cleanup.

Draft 5 [June 1990] First IEEE ballot.

— Mostly clarifications and cleanups from the mock ballot.

— The `pshar` utility was removed.

— The `ps` utility was given a standardized method of producing custom output.

Draft 4 [February 1990] ‘‘Mock ballot’’ conducted by working group members only. During this ballot, numerous changes were made to ensure that the interfaces closely adhered to the existing practice of the Base Documents, with the exceptions noted in the POSIX.2a PAR (and in the Scope).

POSIX.2a Technical Reviewers

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

Table i — POSIX.2a Technical Reviewers

Section	Topic	Reviewer	
1	*Revisions to General*	Jespersen	
2	*Revisions to Terminology and General Requirements*	Jespersen	
3	*Revisions to Shell Command Language*	Jespersen	7
4	*Revisions to Execution Environment Utilities*	Jespersen	7
5	*User Portability Utilities Option:* `patch`	Bostic	8
5	*User Portability Utilities Option: the rest*	Jespersen	7
	Front matter and Annexes	Jespersen	

P1003.2a Proposed Schedule

This section will not appear in the final document. It is used to provide editorial notes regarding the proposed P1003.2a schedule.

Date	Milestone (End of Meeting)	Draft	
Oct 24-28, 1988 *Honolulu, HI*	UPE planning begins.	–	
Jan 9-11, 1989 *Ft. Lauderdale, FL*	Begin UPE command selections and initial definitions.	–	
Apr 24-28, 1989 *Minneapolis, MN*	Select and define UPE utilities; PAR approved by Sponsor Executive Committee	**0**	
Jul 10-14, 1989 *San Jose, CA*	Refine utility descriptions; resolve version control contents.	**1**	
Oct 16-20, 1989 *Brussels, Belgium*	Refine utility descriptions; freeze utility selections; address internationalization issues.	**2**	
Jan 8-12, 1990 *New Orleans, LA*	Prepare Working Group Mock Ballot.	**3**	
[Feb 1990 - Apr 1990]	Working Group Mock Ballot.	**4**	
Apr 23 - 27, 1990 *Salt Lake City, UT*	Resolve Working Group Mock Ballot.	**4**	
[Jul 1990 - Feb 1991]	[Ballot Draft 5]	**5**	
[Mar 1991 - Jun 1991]	[Ballot Draft 6]	**6**	
[Jul 1991 - Oct 1991]	[Ballot Draft 7]	**7**	
[Jan 1992]	[Ballot Draft 8]	**8**	8
[Jan 1992 - May 1992]	[WG15 Balloting of PDAM 9945-2.1]	**8**	8

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

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

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

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of the IEEE , the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, the IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

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

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

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

Contents

TABLES

Introduction

(This Introduction is not a normative part of P1003.2a Information technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment 1: User Portability Extension (UPE), but is included for information only.)

Editor's Note: This Introduction consists of material that will eventually be integrated into the base POSIX.2 standard's Introduction (and the portion of Annex B that contains general rationale about the standard). The Introduction contains text that was previously held in either the Foreword or Scope. As this portion of the standard is for information only (nonnormative), specific details of the integration with POSIX.2 are left as an editorial exercise.

The purpose of this document is to supplement the work begun in P1003.2, Shell and Utility Application Interface for Computer Operating System Environment, to promote the ''portability'' of users and programmers between conforming systems. The original POSIX.2 sought to define a standard interface and environment for application programs that require the services of a ''shell'' command language interpreter and a set of common utility programs. This supplement extends the list of utilities, and features of previously-defined utilities, to provide a common interactive environment. It is intended for end users, program developers, systems implementors, and application software developers and is complementary to ISO/IEC 9945-1: 1990 (first in a family of 'POSIX'' standards), which specifies operating system interfaces and source code level functions, based on the UNIX[1] system documentation; POSIX.2 and POSIX.2a are based upon documentation and the knowledge of existing programs that assume an interface and architecture similar to that described by POSIX.1 {8}.[2]

This document consists of a series of additions to the POSIX.2 base standard. Some are completely new utilities, others are added features or behaviors that are appropriate for the interactive nature of the scope of work. For the immediate future, this document must be considered as a standalone work; it is not intended to be a series of changes that immediately overthrow the base POSIX.2 standard. The intent of the working group is that POSIX.2 and POSIX.2a exist as separate IEEE documents, separately specifiable by the marketplace (although a system cannot claim conformance only to POSIX.2a). However, these two documents may eventually be accepted by ISO/IEC JTC 1 as international standards and it is intended that, in that timeframe (1992 or 1993), POSIX.2 and POSIX.2a be combined into a single integrated IS 9945-2 (although every portion marked as an optional facility, such as the UPE itself, would remain optional).

---

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

2) The number in braces corresponds to those of the references in 1.2 (or the bibliographic entry in Annex A if the number is preceded by the letter B). The normative references clause and bibliography annex appear in POSIX.2, Draft 11.1.

The developers of this standard established the following goals for their work:

Extend/supplement the shell services and common utilities of POSIX.2 for the Portable Operating System Interface for Computer Environments (ISO/IEC 9945-1: 1990) and similar systems. Provide extensions to those services and additional utilities to support terminal users in a consistent manner across all conforming systems. The extension, to be packaged as a supplement to the POSIX.2 standard, is limited by three constraints:

(1) The *users* in this context are limited to the group of individuals who are familiar with the style of interaction characteristic of historically derived systems based on one of the UNIX operating systems. Typical users would include program developers, engineers, or general purpose time-sharing users.

(2) The *environment* to be supported is a multi-user time-sharing system supporting character-oriented display terminals. Alternatively, it is a collection of single-user systems interconnected via local area networks or telephone lines, but with similar user interfaces. The standard does not include support tailored for bit-mapped or graphics display terminals, although it is expected that such terminals could emulate the character orientation required by this environment. When facilities require cursor addressability from the terminal hardware, these shall be specifically identified in the standard.

(3) The facilities to be provided are based on the historical models of the following documents: the *System V Interface Definition* {B24}; the BSD User Manual {B28}; the *X/Open Portability Guide* {B31}; and, documentation for the KornShell {B25}. Emphasis will be placed on standardizing existing practice for existing users, with changes or additions limited to correcting deficiencies in the following areas:

  (a) Support for international character sets and other localization requirements, such as date formats, collation sequences, etc.

  (b) Reconciliation of differences between the historical implementations

  (c) Elimination of system or device dependencies

  (d) Corrections of features that could reduce system or user security/integrity

The extended facilities shall be designed to be cognizant of international character sets and other localization requirements, such as date formats, collation sequences, etc.

The extended facilities to be provided may include:

(1) Enhancements to utilities already defined in POSIX.2 to facilitate interactive use

(2) Utilities for editing and displaying files on full-screen character-oriented display terminals

(3) Utilities for querying and managing the user's environment

(4) Utilities for interacting with other users on the system or local network

(5) Utilities for scheduling and controlling the user's background, or periodically running, work

(6) Utilities to enhance the interactive program development activities, complementary to those tools used for software installation in the original project

(7) Other user portability utilities and features within the general limitations described by this Introduction

The following areas were considered to be outside the scope of this standard:

(1) Operating system administrative utilities (privileged processes, system processes, daemons, etc.)

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

(3) Networking commands

(4) Graphics programs or interfaces

(5) Text formatting programs or languages

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

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

This section summarizes the deliberations of the IEEE P1003.2 working group, the committee charged by the IEEE Computer Society's Technical Committee on Operating Systems and Operational Environments with devising an interface standard for supplementing the original POSIX.2 standard for user portability.

The rationale annex is being published along with the standard to assist in the process of review. It contains historical information concerning the contents of the standard and why features were included or discarded by the working group. It also contains notes of interest to application programmers on recommended programming practices, emphasizing the consequences of some aspects of the standard that may not be immediately apparent.

The purpose of the original project was to extend the services of the POSIX.1 {8} standard that were available to the portable applications program. The extended services consisted of a shell command language and a collection of utility programs (tools) that could be called from applications for complex data and file manipulation; both were to be based on the same historical models as the POSIX.1 {8} standard.

Draft 2 concluded the scope statement with: ''This supplement requires a fully conforming POSIX.2 base.'' The sentence was removed because when the revised material is integrated into the base standard, it should be perfectly obvious that it is not meant to be a stand alone system, but merely a series of extensions to the base.

The list of areas outside the scope of this standard is identical to that in the Introduction to POSIX.2, with the exception of the item:

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

This is, of course, the entire purpose of this supplement.

Just as this standard relies on the knowledge of architecture, history, and definitions from the POSIX.1 {8}, this supplement relies on the entirety of POSIX.2. The reader is referred to the Rationale and Notes and Bibliography annexes of POSIX.1 {8} and POSIX.2 for background material about UNIX systems in general and POSIX specifically, which will not be duplicated here.

There was periodic debate in the working group about how various of the standard utilities could be implemented, given the set of features provided by POSIX.1 {8}, the presumed base operating system. For example, the `nice` utility is not necessarily supported by a *nice*() system call or function because one does not exist in POSIX.1 {8}. Thus, `nice` cannot be a Strictly Conforming POSIX Shell Application; at most it could be a Conforming POSIX Shell Application with Extensions. However, there is no requirement stated in this standard that the standard utilities have to be strictly conforming programs themselves. The base standard expressed a goal that: ‘‘This interface should be implementable on conforming POSIX.1 {8} systems.’’ Most, if not all, of the standard utilities in the base standard met that goal, but the expanded scope for this supplement made that more difficult to achieve. A few of the utilities have some requirements for administrative support that were outside the scope of POSIX.1 {8}: `crontab`, `df`, `nice`, `ps`, `renice`, and `who` are examples. It is anticipated that the emerging POSIX System Administration work will provide standard underpinnings for such services, but until they are approved, common historical implementation extensions will have to be relied upon.

There was a number of differences in philosophy used in preparing the utilities in this supplement from those in the base standard:

— It was assumed that a human was the audience for the output of utilities. Unlike a shell script, the primary audience of the base standard, the human is not so concerned about exactness of format. Therefore, in many cases, the UPE merely suggests the form of the output, rather than using the *printf*()-like form of the base standard. This has the benefit of allowing multiple historical implementations to conform without change. There are exceptions to this general philosophy, however. In some cases, such as `df`, it was felt that users were confused by the historical differences in output. A compromise ‘‘portable’’ output format that could be provided by all systems in an upward-compatible manner was provided. That this output could then be parsed by a shell script is a serendipitous side effect. (As explained in the original base standard, `df` is a utility that produces output requiring human judgment for its interpretation; shell scripts should probably avoid using `df` to make decisions about allocating new files.)

— Since the idea of the UPE was to form a common environment for system users of existing technology terminals, and not blaze new trails in user interfaces, the working group chose to limit changes to command names and syntax. While the application orientation of the base standard caused various name and option changes for consistency and historical harmonization, this standard chose to leave many of the unusual quirks intact so that users would continue to feel comfortable at their terminals. It is for this reason that the UPE has comparatively fewer obsolescent features than the base standard does.

Related Standards Activities

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

The following areas are under active consideration at this time, or are expected to become active in the near future: [3)]

(1) Language-independent service descriptions of POSIX.1 {8}

(2) C, Ada, and FORTRAN Language bindings to (1)

(3) Verification testing methods

(4) Realtime facilities

(5) Secure/Trusted System considerations

(6) Network interface facilities

(7) System Administration

(8) Graphical User Interfaces

(9) Profiles describing application- or user-specific combinations of Open Systems standards for: supercomputing, multiprocessor, and batch extensions; transaction processing; realtime systems; and multiuser systems based on historical models

(10) An overall guide to POSIX-based or related Open Systems standards and profiles

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

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

If you have interest in participating in the TCOS working groups addressing these issues, please send your name, address, and phone number to the Secretary, IEEE Standards Board, Institute of Electrical and Electronics Engineers, Inc., P.O. Box

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

1331, 445 Hoes Lane, Piscataway, NJ 08855-1331, and ask to have this forwarded to the chairperson of the appropriate TCOS working group. If you have interest in participating in this work at the international level, contact your ISO/IEC national body.

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

Technical Committee on Operating Systems and Application Environments (TCOS)

Chair: Jehan-François Pâris

TCOS Standards Subcommittee

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

1003.2a Working Group Officials

Chair:	Hal Jespersen
Vice Chair:	Donald W. Cragun
Editor:	Hal Jespersen
Secretaries:	Helene Armitage (1988-1990)
	Dave Grindeland (1991)

Technical Reviewers

Helene Armitage	Dave Grindeland	Gary Miller
Keith Bostic	Hal Jespersen	Scott Sutter
Donald W. Cragun		

Working Group

Helene Armitage	Dave Grindeland	Daniel Nissen
Stephen R. Bartels	Jerry Gross	Landon Curt Noll
Charles C. Bennett, Jr.	Brian Hackworth	Jim Oldroyd
Philippe Bertrand	Michael J. Hannah	John Peace
Ron Bird	Marjorie E. Harris	Per Pedersen
Robert Bismuth	Hiroki Higuchi	Garth Perry
John Bissell	David F. Hinnant	Pat Philip
Jim Blondeau	Gail Holmes	Rob Pike
James C. Bohem	Ron Holt	John Quarterman
Kathy Bohrer	Randall Howard	Joe Ramus
Keith Bostic	Akitaka Iwata	Mike Ressler
Phyllis Eve Bregman	Hal Jespersen	Andrew K. Roach

Peter Brouwer
F. Lee Brown, Jr.
Jonathan Brown
Steve Carter
Tim Carter
Michael W. Cavins
John Caywood
Gerald Charlwood
Mark Colburn
Donald W. Cragun
Claudia DeBlauu
Dave Decot
Mark Doran
Stan Douglas
Edna Edelman
Mike Edmonds
Ron Elliott
Richard W. Elwood
Hirsaki Eto
Ken Faubel
Glenn Fowler
Curtis Galloway
Gary A. Gaudet
Al Gettier
Timothy D. Gill
Gregory Goddard
Loretta Goudie
Mike Karels
Ronald S. Karr
Martin Kirk
Will Kling
Hiromichi Kogure
David Korn
Rick Kuhn
Mike Lambert
Maggie Lee
Greger Leijonhufvud
Bob Lenk
Mark Levine
Dan Magenheimer
Robert J. Makowski
Shane P. McCarron
John McGrory
Stuart McKaig
T. Ted McMillan
Sunil Mehta
Francoise Miane
Bill Middlecamp
Gary W. Miller
Carlos E. Montero-Luque
Yasushi Nakahara
Martha Nalebuff
Parul S. Nanavati
Sonya D. Neufer
Mark Roseman
Piyanai Saowarattitada
Mike Schultz
Glen Seeds
Karen Sheaffer
Harry Singh
James Soddy
Carolyn Surprenant
Scott A. Sutter
Ravi Tavakley
Marc Teitelbaum
Donn Terry
Teoman Topcubasi
Eugene Tsuno
Yarko Tymcuerak
Geraldine Vitovitch
Mike Wallace
Alan Weaver
Larry Wehr
N. Ray Wilkes
David Willcox
David Williams
Neil Winton
Morten With
John Wu
Jim Young

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

Richard Alexander	*SHARE Institutional Representative*
Ralph Barker	*UniForum Institutional Representative*
Michael Lambert	*X/Open Institutional Representative*
John Quarterman	*Usenix Institutional Representative*

Gary M. Allen
Helene Armitage
Charles R. Arnold
David Athersych
Randall Atkinson
Ted P. Baker
Timothy Baker
Ralph Barker
John Barr
Andy Bihain
Robert Bismuth
Nawaf Bitar
George Bittner
James Bohem
Kathryn Bohrer
Keith Bostic
Sven Brehmer
David Brower
Karl Heubaum
David F. Hinnant
Rand Hoven
Randall Howard
Irene Hu
Hai Huang
Richard Hughes-Rowlands
William Hullsiek
Aron K. Insinga
Jim Isaak
Richard James
Hal Jespersen
Lowell Johnson
Michael Jones
Mike Kamrad
Michael J. Karels
Michael Kearney
Judy S. Kerner
Landon Curt Noll
Fred Noz
John Peace
Pat Philip
Chris Phillips
Dave Plauger
Scott E. Preece
Joseph Ramus
Carol Raye
Christopher J. Riddick
Andrew K. Roach
Arnold Robbins
Seth Rosenthal
Piyanai Saowarattitada
John Sauter
Richard Schaaf
Lorne H. Schachter
Carl Schmiedekamp

Gretchen Brown
John Carmichael
Steven Carter
Jerry Cashin
John Caywood
Siddhartha Chatterjee
Andy Cheese
Janice Chelini
Kilnam Chon
Robert L. Claeson
William M. Corwin
Mike Cossey
William Cox
Donald Cragun
Ana Maria DeAlvare
Virgil Decker
Steven Deller
Terence Dowling
Stephen A. Dum
John D. Earls
Edna B. Edelman
Philip H. Enslow
Kester Fong
Sheila Frankel
James Gibson
Michel Gien
Tom Griest
Dave Grindeland
Robert C. Groman
Judy Guist
Gregory Guthrie
Mark Guzzi
Charles Hammons
Carol J. Harkness
Myron Hecht
Hans H. Heilborn
Lorraine C. Kevra
Sandeep Khanna
Paul J. King
M. J. Kirk
David Kirshen
Philip Klimbal
Kenneth Klingman
Joshua Knight
Robert Knighter
Ronnie Kon
Richard Kuhn
Robin B. Lake
Doris Lebovits
Maggie Lee
Robert Lenk
David Lennert
Donald Lewine
Kevin Lewis
F. C. Lim
Warren E. Loper
Joseph F. P. Luhukay
Robert Makowski
William Mar
Roger Martin
Marshall McKusick
Robert McWhirter
Gary W. Miller
James Moe
Carlos Enrique Montero-Luque
James W. Moore
Anita Mundkur
Narendran Nachiappan
Martha Nalebuff
Barry Needham
Daniel Nissen
Richard Scott
Leonard Seagren
Glen Seeds
Richard Seibel
Dan Shia
Jacob Slonim
Charles Smith
Michael Struerwalt
Keith Stuck
Scott Sutter
Ravi Tavakley
Donn S. Terry
Gary F. Tom
Barry Traylor
Mark-Rene Uchida
L. David Umbaugh
Steven M. Valentine
Michael W. Vannier
Dalibor Vrsalovic
Stephen Walli
Tom Watson
Alan Weaver
Mary Weeks
Larry Wehr
Bruce Weiner
Andrew Wheeler, Jr.
Nicholas Ray Wilkes
David Willcox
David Williams
John R. Williams
David Wong
James Young
Oren Yuen
Janusz Zalewski
John Zolnowsky

When the IEEE Standards Board approved this standard on *<date to be provided>* it had the following membership:

(to be pasted in by IEEE)

Information technology — Portable Operating System Interface (POSIX) — Part 2: Shell and Utilities — Amendment 1: User Portability Extension (UPE)

Section 1: Revisions to General

⇒ **1.1 Scope.** *Modify the contents of clause 1.1, Scope, as shown below.*

1.1 Scope

This standard defines a standard source code level interface to command interpretation, or ''shell,'' services and common utility programs for application programs. These services and programs are complementary to those specified by ISO/IEC 9945-1: 1990 {8}, hereinafter referred to as 'POSIX.1 {8}.'' When the User Portability Utilities Option is included, this standard has additional scope. The list of utilities and features is extended to provide a common interactive environment for system users and program developers.

The standard has been designed to be used by both application programmers and system implementors. When the User Portability Utilities Option is included, it is also to be used by system users and program developers. However, it is intended to be a reference document and not a tutorial on the use of the services, the utilities, or the interrelationships between the utilities.

The emphasis of this standard without the User Portability Utilities Option is on the shell and utility functionality required by application programs (including ‘‘shell scripts’’) and not on the direct interactive use of the shell command language or the utilities by humans. When the User Portability Utilities Option is included, the emphasis is extended to support terminal users in a consistent manner across all conforming systems. There are three constraining factors that limit this user portability scope:

(1) The *users* in this context are limited to the group of individuals who are familiar with the style of interaction characteristic of historically derived systems based on one of the UNIX operating systems. Typical users would include program developers, engineers, or general purpose time-sharing users.

(2) The *environment* to be supported is a multi-user time-sharing system supporting character-oriented display terminals. Alternatively, it is a collection of single-user systems interconnected via local area networks or telephone lines, but with similar user interfaces. This standard does not include support tailored for bit-mapped or graphics display terminals, although it is expected that such terminals could emulate the character orientation required by this environment. When facilities require cursor addressability from the terminal hardware, these are specifically identified in the standard.

(3) The facilities to be provided are based on the historical models of the following documents: the *System V Interface Definition* {B24}; the BSD User Manual {B28}; the *X/Open Portability Guide* {B31}; and, documentation for the KornShell {B25}. Emphasis will be placed on standardizing existing practice for existing users, with changes or additions limited to correcting deficiencies in the following areas:

   (a) Support for international character sets and other localization requirements, such as date formats, collation sequences, etc.;

   (b) Reconciliation of differences between the historical implementations;

   (c) Elimination of system or device dependencies;

   (d) Corrections of features that could reduce system or user security/integrity.

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

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

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

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

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

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

The scope was carefully worded to exclude the development of interfaces for new technology environments, such as bit-mapped graphics windowing terminals, and ‘‘look-and-feel’’ considerations (icons, menus, mice, etc.) for novice or specialized system users. There are numerous other standards groups in this domain and POSIX.2 did not want to overlap, but merely satisfy historical users.

The acknowledgment of four base documents placed the UPE effort squarely within guidelines established by the developers of the standard (the IEEE Technical Committee on Operating Systems and Operational Environments) to anchor proposed standards on industry practice and documentation. Since a primary purpose was to carry users familiar with historical practice along into a more portable open systems environment, the base documents were closely adhered to in terms of utility names, features, and behaviors. Only the four exception criteria called for any modifications to historical practice.

Selecting the KornShell documentation was a clear signal by the working group 8
that `ksh` extended scripting features and interactive command-line editing 8
features were major factors in selecting POSIX shell behavior. The System V shell 8
and the C-shell are also in the base documents, but numerous working group dis- 8
cussions resulted in the following guidelines used to prepare and ballot the drafts: 8

- — Use the System V shell as the broad shell scripting base, but select KornShell features to correct historical deficiencies (such as the lack of easy arithmetic and substring manipulation, the tilde expansion, and the problems of command substitution nesting).
- — Use the KornShell model for interactive features such as command-line editing.
- — When required features appeared in both the C-shell and KornShell, emphasize the KornShell interface unless it was clearly deficient. This decision was predicated on the fact that it made less sense to select scripting features and interactive features from two very different shells. Historical C-shell users would find the POSIX shell to be rather different than their experience and selecting isolated C-shell features would only serve to confuse both C-shell and KornShell users.

1.3.1.3 Conforming Implementation Options

⇒ **1.3.1.3 Conforming Implementation Options.** *Modify the contents of subclause 1.3.1.3, to add the following table entries in the proper sorted order:*

{POSIX 2_CHAR _TERM }	The system supports at least one terminal type capable of all operations described in this standard. See 2.14.
{POSIX 2_UPE }	The system supports the User Portability Utilities Option in Section 5.

Conforming Implementation Options Rationale. *(This subclause is not a part of P1003.2a)*

See the rationale for 2.14.

Section 2: Revisions to Terminology and General Requirements

2.2.1 Terminology

⇒ **2.2.1 Terminology.** *Modify the contents of subclause 2.2.1, Terminology, to add the following definition in the correct sorted order [disregarding the subclause number shown here].*

2.2.1.1 application: When the User Portability Utilities Option is supported, requirements associated with the term *application* shall also be interpreted to include the actions of the user who is interacting with the system by entering shell command language statements from a terminal.

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

The term *application* is not defined in either POSIX.1 {8} or POSIX.2; it is assumed to be a part of general computer science terminology. The definition in this subclause is present to ensure readers understand that the general requirements for applications also apply to interactive users. If, for instance, POSIX.2 requires a Strictly Conforming POSIX.2 Application to use a specific option to a utility, an interactive user would also have to use that option to be considered strictly conforming.

2.2.2 General Terms

⇒ **2.2.2 General Terms.** *Modify the following two subclauses of 2.2.2, General Terms, to add the synonym shown for each:*

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

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

Each session that has established a connection with a controlling terminal has exactly one process group of the session as the foreground process group of that controlling terminal. See POSIX.1 {8}7.1.1.4. [POSIX.1 {8}]

⇒ **2.2.2 General Terms.** *Modify the contents of subclause 2.2.2, General Terms, to add the following definitions in the correct sorted order [disregarding the subclause numbers shown here].*

2.2.2.201 alert: To cause the user's terminal to give some audible or visual indication that an error or some other event has occurred.

When the standard output is directed to a terminal device, the method for alerting the terminal user is unspecified. When the standard output is not directed to a terminal device, the alert shall be accomplished by writing the <alert> character to standard output (unless the utility description indicates that the use of standard output produces undefined results in this case).

2.2.2.202 block-mode terminal: A terminal device operating in a mode incapable of the character-at-a-time input and output operations described by some of the standard utilities.

See 2.14.

2.2.2.203 cursor position: The line and column position on the screen denoted by the terminal's cursor.

2.2.2.204 display: To output to the user's terminal.

If the output is not directed to a terminal, the results are undefined.

2.2.2.205 job: A set of processes, comprising a shell pipeline, and any processes descended from it, that are all in the same process group.

See the definition of *pipeline* in 3.9.2.

2.2.2.206 job control job ID: A handle that is used to refer to a job.

The job control job ID can be any of the forms shown in Table 2-100.

Table 2-100 – Job Control Job ID Formats

Job Control Job ID	Meaning
%%	Current job
%+	Current job
%−	Previous job
%*n*	Job number *n*
%*string*	Job whose command begins with *string*
%?*string*	Job whose command contains *string*

2.2.2.207 mount point: Either the root directory or a directory for which the *st_rdev* field of the POSIX.1 {8} *struct stat* differs from that of its parent directory.

2.2.2.208 refresh: To ensure that the information on the user's terminal screen is up-to-date.

2.2.2.209 screen: A rectangular region of columns and lines on a terminal display.

A screen may be a portion of a physical display device or may occupy the entire physical area of the display device.

2.2.2.210 scroll: To move the representation of data vertically or horizontally relative to the terminal screen.

There are two types of scrolling:

(1) The cursor moves with the data

(2) The cursor remains stationary while the data moves

2.2.2.211 suspended job: A job that has received a SIGSTOP, SIGTSTP, SIGTTIN, or SIGTTOU signal that caused the process group to stop.

A suspended job is a background job, but a background job is not necessarily a suspended job.

2.2.2.212 system scheduling priority: A number used as advice to the system to alter process scheduling priorities.

Raising the value should give a process additional preference when scheduling a process to run. Lowering the value should reduce the preference and make a process less likely to run. Typically, a process with higher system scheduling priority will run to completion more quickly than an equivalent process with lower system scheduling priority. A scheduling priority of zero specifies the default policy of the system.

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

This standard assigns precise requirements for the terms *display* and *write,* unlike some historical user manuals. (The term *write* has been moved into the POSIX.2 base standard as of Draft 11.) Some historical systems have chosen to implement certain utilities without using the traditional UNIX system file descriptor model. For example, the `vi` editor might employ direct screen memory updates on a personal computer, rather than a *write* () system call. An instance of user prompting might appear in a little ''pop-up'' dialog box, rather than with standard error. When the standard uses the term *display,* the method of outputting to the terminal is unspecified; many historical implementations use *termcap* or *terminfo* , but this is not a requirement. The term *write* is used when the standard mandates that a file descriptor be used and that the output can be redirected. However, it is assumed that when the writing is directly to the terminal (it has not been redirected elsewhere), there is no practical way for a user or test suite to determine whether a file descriptor is actually being used or not. Therefore, the use of a file descriptor is mandated only for the redirection case and the implementation is free to use any method when the output is not redirected. In previous drafts, there were the terms *print* and *type,* which attempted to exert a finer degree of specification on output and input differences for terminals versus other types of files. These verbs were discarded when it was realized that there is no practical or useful way to determine how I/O is accomplished when it is not redirected. *Read* and *write* are used almost exclusively, with the very few exceptions of those utilities where output redirection need not be supported: `tabs`, `talk`, `tput`, and `vi`.

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

The definition of *cursor* is assumed to be a standard in computer science.

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

The definition of *system scheduling priority* is not intended to suggest that all 7
processes in a system have priorities that are comparable. Scheduling policy 7

extensions such as the realtime priorities in POSIX.4 make the notion of a single 7
underlying priority for all scheduling policies problematic. Some systems may 7
implement the `nice`-related features to affect all processes on the system, others 7
to affect just the general time-sharing activities implied by POSIX.2, and others 7
may have no effect at all. Because of the use of ''implementation defined'' in `nice` 7
and `renice`, a wide range of implementation strategies are possible. 7

2.3 Built-in Utilities

⇒ **2.3 Built-in Utilities.** *Modify the contents of clause 2.3, Built-in Utilities, to add the following utilities to the table of regular built-ins.*

`alias`	`fc`	`jobs`	`unalias`
`bg`	`fg`	`newgrp`	

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

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

2.6 Environment Variables

⇒ **2.6 Environment Variables.** *Modify the contents of clause 2.6, Environment Variables, to add the following variables in the correct sorted order.*

COLUMNS A decimal integer > 0 used to indicate the user's preferred width in column positions for the terminal screen or window. (See the definition of *column position* in 2.2.2.31.) If this variable is unset or null, the implementation determines the number of columns, appropriate for the terminal or window, in an unspecified manner. When **COLUMNS** is set, any terminal-width information implied by **TERM** shall be overridden. Users and conforming applications should not set **COLUMNS** unless they wish to override the system selection and produce output unrelated to the terminal characteristics. This standard specifies the effects of this variable only for the standard utilities defined by the User Portability Utilities Option in Section 5.

LINES A decimal integer > 0 used to indicate the user's preferred number of lines on a page or the vertical screen or window size in lines. A line in this case is a vertical measure large enough to hold the tallest character in the character set being displayed. If this variable is unset or null, the implementation determines the number of lines, appropriate for the terminal or window (size, terminal baud rate, etc.), in an unspecified manner. When **LINES** is set, any terminal-height information implied by **TERM** shall be overridden. Users and conforming applications should not set **LINES** unless they wish to override the system selection and produce output unrelated to the terminal characteristics. This standard specifies the effects of this variable only for the standard utilities defined by the User Portability Utilities Option in Section 5.

Environment Variables Rationale. *(This subclause is not a part of P1003.2a)*

COLUMNS The default value for the number of column positions is unspecified because historical implementations use different methods to determine values corresponding to the size of the screen in which the utility is run. (In Draft 7, all references to **COLUMNS** and **LINES** were made consistent so that the default is unspecified, rather than implementation defined in a few places. Values for these variables that are not decimal integers > 0 are implicitly undefined values; it is unnecessary to enumerate all of the possible values outside of the acceptable set.) This size is typically known to the implementation through the value of **TERM**, or by more elaborate methods such as extensions to the `stty` utility, or knowledge of how the user is dynamically resizing windows on a bit-mapped display terminal. Users should not need to set this variable in the environment unless there is a specific reason to override the implementation's default behavior, such as to display data in an area arbitrarily smaller than the terminal or window. 7

LINES See **COLUMNS**.

2.11.6.1 Standard Output

⇒ **2.11.6.1 Standard Output.** *Modify the contents of subclause 2.11.6.1, Standard Output (part of 2.11, Utility Description Defaults), to add the following paragraph at the end of the subclause:*

When the User Portability Utilities Option is implemented, some of the standard utilities in Section 5 describe their output using the verb *display*, defined in 2.2.2.204. Output described in the Standard Output subclauses of such utilities may be produced using means other than standard output. When standard output is directed to a terminal, the output described shall be written directly to the terminal. Otherwise, the results are undefined.

2.13.2 Symbolic Constants for Portability Specifications

⇒ **2.13.2 Symbolic Constants for Portability Specifications.** *Modify the contents of subclause 2.13.2 to add the following limits in the correct sorted order.*

Name	Description	
{POSIX2_CHAR_TERM}	The system supports at least one terminal type capable of all operations described in this standard. See 2.14. This value need not be present on a system not supporting the User Portability Utilities Option.	7
{POSIX2_UPE}	The system supports the User Portability Utilities Option in Section 5.	

⇒ **B.2.3 Compile-Time Symbolic Constants for Portability Specifications.** *Modify the contents of subclause B.2.3 to add the following limits in the correct sorted order.*

Name	Description	
_POSIX2_CHAR_TERM	The system supports at least one terminal type capable of all operations described in this standard. See 2.14. This value need not be present on a system not supporting the User Portability Utilities Option.	7
_POSIX2_UPE	The system supports the User Portability Utilities Option in Section 5.	

Symbolic Constants Rationale. *(This subclause is not a part of P1003.2a)*

See the rationale for {POSIX2_CHAR_TERM} in 2.14.

In an earlier draft, two limits, {POSIX2_MAX_NICE} and {POSIX2_MIN_NICE}, were included to bound the range of acceptable ‘‘nice values.’’ These limits were removed because `nice` and `renice` were reworded to gracefully accept values outside the range without returning an error.

2.14 Terminal Characteristics

⇒ **2.14 Terminal Characteristics.** *Add a new clause 2.14:*

This clause applies only to systems supporting the User Portability Utilities 7
Option. 7

The standard utilities historically have been implemented on a wide range of terminal types, but a conforming implementation need not support all features of all utilities on every conceivable terminal. This standard states which features are optional for certain classes of terminals in the individual utility description clauses. The implementation shall document which terminal types it supports and which of these features and utilities are not supported by each terminal. This implementation-defined list of terminals:

— Shall include at least one terminal type that is capable of supporting all of the standard utilities and all of their features, if the {POSIX2_CHAR_TERM} option is provided.

— May group terminals in terms of families or equivalences to other documented terminal types.

— Need not consist of an exhaustive list of terminal models when the implementor considers that some terminal types are used too infrequently to be listed.

When a feature or utility is not supported on a specific terminal type, as allowed by this standard, and the implementation considers such a condition to be an error preventing use of the feature or utility, the implementation shall indicate such conditions through diagnostic messages or exit status values or both (as appropriate to the specific utility description) that inform the user that the terminal type lacks the appropriate capability.

This standard uses a notational convention based on historical practice that identifies some of the control characters defined in 2.4 in a manner easily remembered by users on many terminals. The correspondence between this ``control-*char*'' notation and the actual control characters is shown in Table 2-101. When this standard refers to a character by its control- name, it is referring to the actual control character shown in the Value column of the table, which is not necessarily the exact control key sequence on all terminals. Some terminals have keyboards that do not allow the direct transmission of all the nonalphanumeric characters shown. In such cases, the system documentation shall describe which data sequences transmitted by the terminal shall be interpreted by the system as representing the special characters.

NOTE: The notation uses uppercase letters for arbitrary editorial reasons. There is no implication that the keystrokes represent control-shift-letter sequences.

Table 2-101 – Control Character Names

Name	Value	Name	Value	Name	Value
control-A	<SOH>	control-L	<FF>	control-W	<ETB>
control-B	<STX>	control-M	<CR>	control-X	<CAN>
control-C	<ETX>	control-N	<SO>	control-Y	<EM>
control-D	<EOT>	control-O	<SI>	control-Z	<SUB>
control-E	<ENQ>	control-P	<DLE>	control-[	<ESC>
control-F	<ACK>	control-Q	<DC1>	control-\	<FS>
control-G	<BEL>	control-R	<DC2>	control-]	<GS>
control-H	<BS>	control-S	<DC3>	control-^	<RS>
control-I	<HT>	control-T	<DC4>	control-_	<US>
control-J	<LF>	control-U	<NAK>	control-?	<DEL>
control-K	<VT>	control-V	<SYN>		

Terminal Characteristics Rationale. *(This subclause is not a part of P1003.2a)*

There are at least two types of terminals that might have difficulty with certain utility features: hard copy terminals and block-mode terminals.

It is the intention of the developers of this standard that the list of supported terminals and features be presented as tersely as the implementors may desire. Subsets of terminal types may be included, based upon marketing requirements. Generous use of correct global statements is preferable to itemized lists of features for each of hundreds of terminal types. Attention should be directed to clarifying nonsupported features on terminal types that would otherwise be expected to support such features, due to those terminals' similarity to documented types.

It is the intention of the developers of this standard that when a terminal cannot support a certain feature, the utility should produce a useful diagnostic status or message explicitly informing the user of this fact. It should not produce a syntax error, ''unknown command,'' or ''unknown feature'' message or status. (This intent is not in the normative text because it is difficult to describe except with examples.)

Terminals that do not have the ability to send some of the special characters are in wide industry use, although not on the historical systems that form the basis for this standard. For example, the 3270 family of mainframe synchronous terminals cannot directly send the control characters. Therefore, implementations have to document how to do so; it could be by using a series of keystrokes, or by special function keys. In the 3270 example, PF 3 has been used as a `control-D` equivalent.

The {POSIX2_CHAR_TERM} option was added to allow a vendor supporting only block-mode terminals to still conform, even though some utilities, such as `vi`, have limited usefulness. The `vi` editor is the only utility in POSIX.2 that can be entirely omitted if this option is not supported; others, such as `ex`, `more`, and `talk`, still have useful feature subsets that are unrelated to the terminal variety.

The decision on whether to require conformance to {POSIX2_CHAR_TERM} is strictly one of marketing. Even when this option is provided, however, there is no guarantee that *every* terminal on the system will support all features; the option implies that the software drivers and other support features are installed, but not that any of the terminals were actually hooked up.

Section 3: Revisions to Shell Command Language

⇒ **3.1 Shell Definitions.** *Modify the contents of clause 3.1, Shell Definitions, to add the following definition in the correct sorted order [disregarding the subclause number shown here].*

3.1 Shell Definitions

3.1.101 alias name: A word consisting solely of underscores, digits, and alphabetics from the portable character set (see 2.4) and any of the following characters:

`!` `%` `,` `@` 7

Implementations may allow other characters within alias names as an extension. 7

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

The characters allowed in alias names are from historical practice.

⇒ **3.3.1 Alias Substitution.** *Modify the contents of clause 3.3, Token Recognition, to add the following subclause:*

3.3.1 Alias Substitution

The processing of aliases shall be supported if the system supports the User Portability Utilities Option.

After a token has been delimited, but before applying the grammatical rules in 3.10, a resulting word that is identified to be the command name word of a simple command shall be examined to determine if it is an unquoted, valid alias name. However, reserved words in correct grammatical context shall not be candidates for alias substitution. A valid alias name (see 3.1.101) shall be one that has been defined by the `alias` utility (see 5.1) and not subsequently undefined using `unalias` (see 5.31). Implementations also may provide predefined valid aliases that are in effect when the shell is invoked. To prevent infinite loops in recursive aliasing, if the shell is not currently processing an alias of the same name, the word shall be replaced by the value of the alias; otherwise, it shall not be replaced.

If the value of the alias replacing the word ends in a `<blank>`, the shell shall check the next command word for alias substitution; this process shall continue until a word is found that is not a valid alias or an alias value does not end in a `<blank>`.

When used as specified by this standard, alias definitions shall not be inherited by separate invocations of the shell or by the utility execution environments invoked by the shell; see 3.12.

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

The alias capability was added in the UPE because it is widely used in historical implementations by interactive users. It was omitted from the base standard because it is not commonly used in application scripts, particularly since aliases are not passed to child shells.

The definition of *alias name* precludes an alias name containing a slash character. Since the text applies to the command words of simple commands, reserved words (in their proper places) cannot be confused with aliases.

The placement of alias substitution in Token Recognition implies that it precedes all of the word expansion steps.

An example concerning trailing `<blank>`s and reserved words follows. If the user types:

```
$ alias foo="/bin/ls "
$ alias while="/"
```

The effect of executing

```
$ while true
> do
> echo "Hello, World"
> done
```

is a never-ending sequence of `Hello, World` strings to the screen. However, if the user types

```
$ foo while
```

the result will be an `ls` listing of `/`. Since the alias substitution for `foo` ends in a `<space>`, the next word is checked for alias substitution. The next word, `while`, has also been aliased, so it is substituted as well. Since it is not in the proper position as a command word, it is not recognized as a reserved word.

If the user types

```
$ foo; while
```

`while` retains its normal reserved-word properties.

See the rationale for `alias` for a discussion of exportable aliases.

3.5.3 Variables

⇒ **3.5.3 Variables.** *Modify the contents of subclause 3.5.3, Variables, to add the following environment variables in the correct sorted order.*

ENV
This variable, when the shell is invoked, shall be subjected to
parameter expansion (see 3.6.2) by the shell and the resulting
value shall be used as a pathname of a file containing shell com-
mands to execute in the current environment. The file need not
be executable. If the expanded value of **ENV** is not an absolute 7
pathname, the results are unspecified. **ENV** shall be ignored if
the user's real and effective user IDs or real and effective group
IDs are different. This standard specifies the effects of this vari- 7
able only for systems supporting the User Portability Utilities 7
Option. 7

LINENO
This variable shall be set by the shell to a decimal number
representing the current sequential line number (numbered start-
ing with 1) within a script or function before it executes each com-
mand. If the user unsets or resets **LINENO**, the variable may
lose its special meaning for the life of the shell. If the shell is not
currently executing a script or function, the value of **LINENO** is
unspecified. This standard specifies the effects of this variable 7
only for systems supporting the User Portability Utilities Option. 7

PPID
This variable shall be set by the shell to the decimal process ID of
the process that invoked this shell. In a subshell (see 3.12), **PPID**
shall be set to the same value as that of parent of the current
shell. For example, `echo $PPID` and `(echo $PPID)` would pro-
duce the same value. This standard specifies the effects of this 7
variable only for systems supporting the User Portability Utilities 7
Option. 7

PS1
Each time an interactive shell is ready to read a command, the
value of this variable shall be subjected to parameter expansion
and written to standard error. The default value shall be `"$ "`. 7
For users who have specific additional implementation-defined 7
privileges (see 2.2.2.6) the default may be another, 7
implementation-defined, value. The shell shall replace each 7
instance of the character `!` in **PS1** with the history file number of
the next command to be typed. (See 5.12.) Escaping the `!` with 8
another `!` (i.e., `!!`) shall place the literal character `!` in the
prompt. This standard specifies the effects of this variable only 7
for systems supporting the User Portability Utilities Option. 7

PS 2 Each time the user enters a <newline> prior to completing a command line in an interactive shell, the value of this variable shall be subjected to parameter expansion and written to standard error. The default value shall be `"> "`. This standard specifies the effects of this variable only for systems supporting the User Portability Utilities Option.

PS 4 When an execution trace (`set −x`) is being performed in an interactive shell, before each line in the execution trace, the value of this variable shall be subjected to parameter expansion and written to standard error. The default value shall be `"+ "`. This standard specifies the effects of this variable only for systems supporting the User Portability Utilities Option.

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

The following entries represent additional information about variables included in the standard, or rationale for common variables in use by shells that have been excluded.

_ (underscore)
While underscore is historic practice, its overloaded usage in the KornShell is confusing and it has been omitted from POSIX.2.

ENV This can be used to set aliases and other items local to the invocation of a shell. The file referred to by **ENV** differs from `$HOME/.profile` in that `.profile` is typically executed at session startup, whereas the **ENV** file is executed at the beginning of each shell invocation.

The **ENV** value is interpreted in a manner similar to a dot script, in that the commands are executed in the current environment and the file needs to be readable, but not executable. However, unlike dot scripts, no **PATH** searching is performed. This is used as a guard against Trojan Horse security breaches.

ERRNO Omitted from the standard because the values of POSIX.1 {8} error numbers are not defined in a portable manner.

FCEDIT In an earlier draft, **FCEDIT** was included in this subclause. However, since it affects only the `fc` utility, it has been removed from this more global place. The value of **FCEDIT** does not affect the command-line editing mode in the shell; see the description of `set -o vi` in 3.14.11.

LINENO This variable is frequently used in **PS 4** during trace mode (`set −x`).

PPID Without this variable, there is no way for a utility to signal its parent or to find its parent process. This is also useful to know if the shell has been ‘‘orphaned.’’

PS 1 Used for interactive prompts. Historically, the ‘‘super-user’’ has 7
had a prompt of `"# "`. Since privileges are not required to be 7
monolithic, it is difficult to define which privileges should cause 7
the alternate prompt. However, a sufficiently powerful user 7
should be reminded of that power by having an alternate prompt. 7

PS 2 Used for interactive prompts.

PS 3 Used by the KornShell for the `select` command. Since the POSIX. 2 shell does not support `select`, **PS 3** was omitted.

PS 4 Used for shell debugging. For example, the following script:

```
PS4='[${LINENO}]+ '
set −x
echo Hello
```

writes the following to standard error:

```
[3]+ echo Hello
```

RANDOM This pseudo-random number generator was not seen as being very useful to interactive users.

SECONDS Although this variable is sometimes used with **PS 1** to allow the display of the current time in the user’s prompt, it is not one that would be manipulated frequently enough by an interactive user to include in POSIX.

3.12 Shell Execution Environment

⇒ **3.12 Shell Execution Environment.** *Add another item to the list of the contents of the shell execution environment:*

— Shell aliases (see 3.3.1).

3.14.11 `set` — Set/unset options and positional parameters

⇒ **3.14.11 `set`.** *Merge the following options into the the first two synopsis lines of* 7
subclause 3.14.11 with appropriate editorial wording indicating that these 7
options need to be supported only when the UPUO is supported: 7

`set` [`-bm`] [`-o` *option*] [*argument* ...] 7
`set` [`+bm`] [`+o` *option*] [*argument* ...] 7

⇒ **3.14.11 `set`.** *Modify the contents of subclause 3.14.11 to add the following options in the correct sorted order.*

`−b` Cause the shell to notify the user asynchronously of background job completions. The following message shall be written to standard error:

`"[%d]%c %s%s\n"`, *<job-number>* , *<current>* , *<status>*, *<job-name>*

where the fields shall be as follows:

<current>
The character `+` identifies the job that would be used as a default for the `fg` or `bg` utilities; this job can also be specified using the *job_id* `%+` or `%%`. The character `−` identifies the job that would become the default if the current default job were to exit; this job can also be specified using the *job_id* `%–`. For other jobs, this field is a `<space>`. At most one job can be identified with `+` and at most one job can be identified with `−`. If there is any suspended job, then the current job shall be a suspended job. If there are at least two suspended jobs, then the previous job shall also be a suspended job.

<job-number>
A number that can be used to identify the process group to the `wait`, `fg`, `bg`, and `kill` utilities. Using these utilities, the job can be identified by prefixing the job number with `%`.

<status>
Unspecified.

<job-name>
Unspecified.

When the shell notifies the user a job has been completed, it may remove the job's process ID from the list of those ‘‘known in the current shell execution environment’’; see 3.9.3.1. Asynchronous notification shall not be enabled by default.

`−m` This option shall be supported if the system supports both the User Portability Utilities Option and the POSIX.1 {8} Job Control Option. All jobs shall be run in their own process groups. Immediately before the shell issues a prompt after completion of the background job, a message reporting the exit status of the background job shall be written to standard error. If a foreground job stops, the shell shall write a message to standard output to that effect, formatted as described by the `jobs` utility (see 5.15.6.1). In addition, if a job changes status other than exiting (for example, if it stops for input or output, or is stopped by a SIGSTOP signal), the shell shall write a similar message immediately prior to writing the next prompt. On systems supporting the POSIX.1 {8} Job Control Option, this option shall be enabled by default for interactive shells.

`−o` *option*

Set various options, many of which shall be equivalent to the single option letters. The following values of *option* shall be supported:

`allexport` Equivalent to `−a`.

8

`errexit` Equivalent to `−e`.

`ignoreeof` Prevent an interactive shell from exiting on end-of-file. 7 7

`monitor` This option-argument shall be supported if the system supports both the User Portability Utilities Option and the POSIX.1 {8} Job Control Option. It shall be equivalent to the `−m` option.

`noclobber` Equivalent to `−C` (uppercase C).

`noglob` Equivalent to `−f`.

`noexec` Equivalent to `−n`.

`notify` Equivalent to `−b`.

`nounset` Equivalent to `−u`.

`verbose` Equivalent to `−v`.

`vi` Allow shell command-line editing using the built-in `vi` editor. Enabling `vi` mode shall disable any other command-line editing mode provided as an implementation extension. 8 8 8

It need not be possible to set `vi` mode on for certain block-mode terminals.

`xtrace` Equivalent to `−x`.

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

The `ignoreeof` setting prevents accidental logouts when the end-of-file character 8
(typically `control-D`) is entered. A user must explicitly `exit` to leave the 7
interactive shell. 7

The `-o` option was originally in the base POSIX.2 document, but was removed because it is not commonly used by scripts and all of the options required for application portability had (or were given) single-letter option names. However, users have different needs and the `-o` option was adopted from the KornShell to address these. In addition to its generally friendly interface, `-o` is needed to pro-
vide the `vi` command-line editing mode, for which historical practice yields no 8
single-letter option name. (Although it might have possible to invent such a 8
letter, it was recognized that other editing modes would be developed and `-o` pro- 8
vides ample name space for describing such extensions.) 8

It is not possible to use the `$-` special parameter to determine the current setting of either of these two modes. In the KornShell, the *option* option-argument can be omitted, producing a report of current option settings. Since it breaks the utility syntax guidelines, and since the output format was unspecified (it changed between KornShell versions), this usage was omitted.

The `set -m` option was added to apply only to the UPE because it applies primarily to interactive use, not shell script applications.

The new `set -o` option-arguments apply only to the UPE because they are needed to enable the line editing modes to be used in the additions to `sh` for the UPE.

The ability to do asynchronous notification became available in the 1988 version of the KornShell. To have it occur, the user had to issue the command

```
trap "jobs -n" CLD
```

The C-shell provides two different levels of an asynchronous notification capability. The environment variable **notify** is analogous to what is done in `set -b` or `set -o notify`. When set, it notifies the user immediately of background job completions. When unset, this capability is turned off.

The other notification ability comes through the built-in utility `notify`. The syntax is

```
notify [%job ... ]
```

By issuing `notify` with no operands, it causes the C-shell to notify the user asynchronously when the state of the current job changes. If given operands, `notify` will asynchronously inform the user of changes in the states of the specified jobs.

To add asynchronous notification to the POSIX shell, neither the KornShell's extensions to `trap`, nor the C-shell's **notify** environment variable seemed appropriate (the **notify** variable is not a proper POSIX environment variable name).

The `set -b` option was selected as a compromise.

The `notify` built-in was considered to have more functionality than was required for simple asynchronous notification.

Section 4: Revisions to Execution Environment Utilities

4.12 **command** — Execute a simple command

⇒ **4.12.1 command Synopsis.** *Add the following lines to the* command *utility Synopsis:*

On systems supporting the User Portability Utilities Option, the following forms shall also be supported:

command [−v | −V] *command_name*

⇒ **4.12.2 command Description.** *Add the following paragraph to the end of the* command *utility Description:*

On systems supporting the User Portability Utilities Option, the command utility shall also provide information concerning how a command name will be interpreted by the shell; see −v and −V.

⇒ **4.12.3 command Options.** *Add the following options in the correct sorted order:*

−v Write a string to standard output that indicates the pathname or command that will be used by the shell, in the current shell execution environment (see 3.12), to invoke *command_name*.

— Utilities, regular built-in utilities, *command_names* including a slash character, and any implementation-provided functions that are found using the **PATH** variable (as described in 3.9.1.1), shall be written as absolute pathnames.

— Shell functions, special built-in utilities, regular built-in utilities not associated with a **PATH** search, and shell reserved words shall be written as just their names.

— An alias shall be written as a command line that represents its alias definition.

— Otherwise, no output shall be written and the exit status shall reflect that the name was not found.

`-V` Write a string to standard output that indicates how the name given in the *command_name* operand will be interpreted by the shell, in the current shell execution environment (see 7
3.12). Although the format of this string is unspecified, it 7
shall indicate in which of the following categories *command_name* falls and include the information stated:

— Utilities, regular built-in utilities, and any implementation-provided functions that are found using the **PATH** variable (as described in 3.9.1.1), shall be identified as such and include the absolute pathname in the string.

— Other shell functions shall be identified as functions.

— Aliases shall be identified as aliases and their definitions shall be included in the string.

— Special built-in utilities shall be identified as special built-in utilities.

— Regular built-in utilities not associated with a **PATH** 7
search shall be identified as regular built-in utilities. (The 7
term ''regular'' need not be used.) 7

— Shell reserved words shall be identified as reserved words.

⇒ **4.12.6.1 `command` Standard Output.** *Change* 'None. *''to:*

When the `-v` option is specified, standard output shall be formatted as:

`"%s\n"`, <*pathname or command*>

When the `-V` option is specified, standard output shall be formatted as:

`"%s\n"`, <*unspecified*>

⇒ **4.12.8 `command` Exit Status.** *Change the first sentence, beginning* 'The command utility shall exit ... *''to*

When the `-v` or `-V` options are specified, the `command` utility shall exit with 8
one of the following values:

0 Successful completion.

>0 The *command_name* could not be found or an error occurred.

Otherwise, the `command` utility shall exit with one of the following values:

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

The `command` –v and –V options were added to the UPE to satisfy requirements from users that are currently accomplished by three different historical utilities: `type` in the System V shell, `whence` in the KornShell, and `which` in the C-shell. Since there is no historical agreement on how and what to accomplish here, the POSIX `command` utility was enhanced and the historical utilities were left unmodified. The C-shell `which` merely conducts a path search. The KornShell `whence` is more elaborate—in addition to the categories required by POSIX, it also reports on tracked aliases, exported aliases, and undefined functions.

Since the –v and –V options of `command` produce output in relation to the current 7
shell execution environment, `command` is generally provided as a shell regular 7
built-in. If it is called in a subshell or separate utility execution environment, 7
such as one of the following: 7

```
(PATH=foo command -v)
nohup command -v
```
7
7

it will not necessarily produce correct results. For example, when called with 7
`nohup` or an *exec* function, in a separate utility execution environment, most 7
implementations will not be able to identify aliases, functions, or special built-ins. 7

Two types of regular built-ins could be encountered on a system and these are 7
described separately by `command`. The description of command search in 3.9.1.1 7
allows for a standard utility to be implemented as a regular built-in as long as it 7
is found in the appropriate place in a **PATH** search. So, for example, 7
`command -v true` might yield `/bin/true` or some similar pathname. Other 7
implementation-provided utilities that are not defined by POSIX.2 might exist only 7
as built-ins and have no pathname associated with them. These will produce out- 7
put identified as (regular) built-ins. Applications encountering these will not be 7
able to count on *exec*ing them, using them with `nohup`, overriding them with a 7
different **PATH**, etc. 7

The output format of –V was left mostly unspecified because human users are its only audience. Applications should not be written to care about this information; they can use the output of –v to differentiate between various types of commands, but the additional information that may be emitted by the more verbose –V is not needed and should not be arbitrarily constrained in its verbosity or localization for application parsing reasons.

4.32 `kill` — Terminate or signal processes

⇒ **4.32.4 `kill` Operands.** *In clause 4.32, modify the description of the pid operand in the Operands subclause:*

pid One of the following:

(1) *<current contents>*

(2) On systems supporting both the POSIX.1 {8} job control option and the User Portability Utilities Option, a job control job ID (see 2.2.2.206) that identifies a background process group to be signaled. The job control job ID notation is applicable only for invocations of `kill` in the current shell execution environment; see 3.12.

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

The job control job ID notation is not required to work as expected when `kill` is operating in its own utility execution environment. In either of the following examples:

```
nohup kill %1 &
system("kill %1");
```

the `kill` operates in a different environment and will not share the shell's understanding of job numbers.

4.40 `mailx` — Process messages

Editor's Note: This clause contains text that is directly extracted from POSIX. 2 8
Draft 11.2 and, as such, is not eligible for balloting by the POSIX. 2a group. This 8
text is marked with the '1'' or '2'' diff-marks. 8

⇒ **4.40 `mailx`.** *Replace the entire contents of clause 4.40 with:*

4.40.1 Synopsis

Send Mode:

`mailx [−s` *subject* `]` *address* ...

On systems supporting the User Portability Utilities Option, the following forms shall also be valid (Receive Mode):

`mailx −e`

`mailx [−HiNn] [−u` *user* `]` 7

`mailx −f [−HiNn] [`*file* `]`

4.40.2 Description

The `mailx` utility provides a message sending and receiving facility.

When this standard is implemented without the User Portability Utilities Option,
an application using `mailx` shall have the ability to send messages directly from
standard input to one or more addresses in an unspecified manner. (Send Mode.) 1
Unless the first character of one or more lines is tilde (~), all characters in the 1
input message shall appear in the delivered message, but additional characters 1
may be inserted in the message before it is retrieved. 1

On systems supporting the User Portability Utilities Option, mail-receiving capabilities and other interactive features, Receive Mode, described below, shall also be enabled.

When reading mail, `mailx` provides commands to facilitate saving, deleting, and responding to messages. When sending mail, `mailx` allows editing, reviewing, and other modification of the message as it is entered.

Incoming mail shall be stored in one or more unspecified locations for each user, 7
collectively called the system *mailbox* for that user. When `mailx` is invoked in 7
Receive Mode, the system mailbox shall be the default place to find them. As
messages are read, they shall be marked to be moved to a secondary file for
storage, unless specific action is taken. This secondary file is called the *mbox* and 7
is normally located in the **HOME** directory of the user (see **MBOX**, in 4.40.5.3, for
a description of this file). Messages shall remain in this file until explicitly

removed. When the `−f` option is used to read mail messages from secondary files, messages shall be retained in those files unless specifically removed. All three of these locations—system mailbox, *mbox*, and secondary file—are referred to in this clause as simply ‘‘mailboxes,’’ unless more specific identification is required.

4.40.3 Options

The `mailx` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation. Only the `−s` *subject* option shall be required on all systems. The other options are required only on systems supporting the User Portability Utilities option.

`−e` Test for the presence of mail in the system mailbox. The `mailx` 7
utility shall write nothing and exit with a successful return code if there is mail to read.

`−f` Read messages from the file named by the *file* operand instead of the system mailbox. (See also `folder`). If no *file* operand is specified, read messages from the *mbox* instead of the system mailbox.

`−H` Write a header summary only.

`−i` Ignore interrupts. (See also `ignore`).

`−n` Do not initialize from the system default startup file. See 4.40.7.

`−N` Do not write an initial header summary.

`−s` *subject* Set the Subject header field to *subject*. All characters in the *sub-* 2
ject string shall appear in the delivered message. The results are 2
unspecified if *subject* is longer than {LINE_MAX} − 10 bytes or con- 2
tains a `<newline>`. 2

`−u` *user* Read the system mailbox of the login name *user*. This shall only
be successful if the invoking user has the appropriate privileges 7
to read the system mailbox of that user.

4.40.4 Operands

The following operands shall be supported by the implementation:

address Addressee of message. When `−n` is specified and no user startup 7
files are accessed (see 4.40.7), this shall be an address to pass to 7
the mail delivery system. Any system or user startup files may 7
enable aliases (see 4.40.7.2.1), that may modify the form of 7
address before it is passed to the mail delivery system. 7

file A pathname of a file to be read instead of the system mailbox
when `-f` is specified. The meaning of the *file* option-argument 8
shall be affected by the contents of the `folder` internal variable; 8
see 4.40.7.1. 8

4.40.5 External Influences

4.40.5.1 Standard Input

When `mailx` is invoked in Send Mode (the first synopsis line), standard input shall be the message to be delivered to the specified addresses. If the User Portability Utilities Option is not supported, standard input lines beginning with a tilde (~) character produce unspecified results.

When `mailx` is invoked in Receive Mode, (the second to fourth synopsis lines), standard input can contain escaped commands (by default, lines beginning with tilde), as described in 4.40.7.

4.40.5.2 Input Files

When `mailx` is used as described by this standard, the *file* option-argument (see 7
the `-f` option) and the *mbox* shall be text files containing mail messages, format- 7
ted as described in 4.40.6.3. The nature of the system mailbox is unspecified; it 7
need not be a file. 7

4.40.5.3 Environment Variables

The following environment variables shall affect the execution of `mailx`:

DEAD This variable shall be interpreted as a pathname of the file in which to save partial messages in case of interrupts or delivery errors. The default shall be `dead.letter` in the directory named by the **HOME** variable. The behavior of `mailx` in saving partial messages is unspecified if the User Portability Utilities Option is not supported and
DEAD is not defined with the value `/dev/null`. 7

EDITOR This variable shall be interpreted as a name of a utility to invoke when the `edit` (see 4.40.7.2.8) or `~e` (see 4.40.7.3),
command is used. The default is unspecified. This stan- 7
dard specifies the effects of this variable only for systems 7
supporting the User Portability Utilities Option. 7

HOME This variable shall be interpreted as the pathname of the user's home directory.

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files) and the handling of case-insensitive address and header-field comparisons.

LC_MESSAGES This variable shall determine the language in which messages should be written.

LC_TIME This variable may determine the format and contents of 7
the date and time strings written by `mailx`. This stan- 7
dard specifies the effects of this variable only for systems 7
supporting the User Portability Utilities Option. 7

LISTER This variable shall be interpreted as a string representing the command for writing the contents of the `folder` directory to standard output when the `folders` command is given (see 4.40.7.2.11). Any string acceptable as a *command_string* operand to the `sh -c` command (see 4.56) shall be valid. If this variable is null or not set, the output command shall be `ls` (see 4.39). The default value
shall be unset. This standard specifies the effects of this 7
variable only for systems supporting the User Portability 7
Utilities Option. 7

MAILRC This variable shall be interpreted as a pathname of the startup file. The default shall be `.mailrc` in the **HOME** directory. The behavior of `mailx` is unspecified if the User Portability Utilities Option is not supported and
MAILRC is not defined with the value `/dev/null`. 7

MBOX This variable shall be interpreted as a pathname of the file to save messages from the system mailbox that have been read. The `exit` command shall override this function, as shall saving the message explicitly in another file. The default shall be `mbox` in the directory named by the
HOME variable. This standard specifies the effects of this 7
variable only for systems supporting the User Portability 7
Utilities Option. 7

PAGER This variable shall be interpreted as a string representing an output filtering and/or pagination command for writing the output to the terminal. Any string acceptable as a *command_string* operand to the `sh -c` command (see 4.56) shall be valid. When standard output is a terminal device, the message output shall be piped through the command if the `mailx` internal variable `crt` is set to a value less the number of lines in the message; see 4.40.7.1. If the **PAGER** variable is null or not set, the paginator shall
be either `more` (see 5.18) or another paginator utility 8
documented in the system documentation. This standard 8
specifies the effects of this variable only for systems sup- 7
porting the User Portability Utilities Option. 7

SHELL The name of a preferred command interpreter. The
default shall be `sh` (see 4.56). This standard specifies the 7
effects of this variable only for systems supporting the 7
User Portability Utilities Option. 7

TERM This variable shall be interpreted as the name of the ter-
minal type, to determine in an unspecified manner, if the 7
internal variable `screen` is not specified, the number of 7
lines in a screenful of headers. If **TERM** is not set or is set 7
to null, an unspecified default terminal type shall be used 7
and the value of a screenful is unspecified. This standard 7
specifies the effects of this variable only for systems sup- 7
porting the User Portability Utilities Option. 7

VISUAL This variable shall be interpreted as a pathname of a utility to invoke when the `visual` command (see 4.40.7.2.38) or ~`v` command-escape (see 4.40.7.3), is used. If this variable is null or not set, the full-screen editor shall be `vi`
(see 5.35). This standard specifies the effects of this vari- 7
able only for systems supporting the User Portability Util- 7
ities Option. 7

4.40.5.4 Asynchronous Events

When `mailx` is in Send Mode and standard input is not a terminal, it shall take the standard action for all signals; see 2.11.5.4.

In Receive Mode, or in Send Mode when standard input is a terminal, if a SIGINT signal is received:

(1) If in command mode, the current command, if there is one, shall be aborted, and a command-mode prompt shall be written.

(2) If in input mode:

   (a) If `ignore` is set, the interrupt shall be ignored, and processing continues.

(b) If the interrupt was received while sending mail, either when in Receive Mode or in Send Mode, a message shall be written, and
another subsequent interrupt, with no other intervening characters 7
typed, shall be required to abort the mail message. If in Receive 7
Mode and another interrupt is received, a command-mode prompt shall be written. If in Send Mode and another interrupt is received, `mailx` shall terminate with a nonzero status.

In both cases listed in (b), if the message is not empty:

[1] If `save` is enabled and the file named by **DEAD** can be created, the message shall be written to the file named by **DEAD** . If the file exists, the message shall be written to replace the contents of the file.

[2] If `save` is not enabled, or the file named by **DEAD** cannot be created, the message shall not be saved.

The `mailx` utility shall take the standard action for all other signals; see 2.11.5.4.

4.40.6 External Effects

4.40.6.1 Standard Output

In command and input modes, all output, including prompts and messages, shall be written to standard output.

4.40.6.2 Standard Error

Used only for diagnostic messages.

4.40.6.3 Output Files

Various `mailx` commands and command escapes can create or add to files, including the *mbox* , the dead-letter file, and secondary mailboxes. When `mailx` is used 7
as described in this standard, these files shall be text files, formatted as follows: 7

line beginning with `"From "` 7
[*header-line(s); see 4.40.7.2*] 8
empty line 7
[*body line(s)* 7
empty line] 7
[*line beginning with* `"From "` ...] 8

where each message begins with the `"From "` line shown, preceded by the begin- 8
ning of the file or an empty line. (The `"From "` line is considered to be part of the 8
message header, but not one of the ‘‘header-lines’’ referred to in 4.40.7.2; thus, it 8
is not affected by the `discard`, `ignore`, or `retain` commands). The formats of 8
the remainder of the `"From "` line and any additional header lines are 7
unspecified, except that none shall be empty. The format of a message body line 7

is also unspecified, except that none shall start with `"From "`; `mailx` shall modify any user-entered message body lines beginning with `"From "` by adding one or more characters to precede the `F`.

When a message from the system mailbox or entered by the user is not a text file, it is implementation defined how such a message is stored in files written by `mailx`.

4.40.7 Extended Description

The entire Extended Description subclause (4.40.7 through 4.40.7.3) shall apply only to implementations supporting the User Portability Utilities option.

When `mailx` is invoked using one of the Receive Mode synopsis forms, it shall write the page of header-summary lines (see below) containing the first new message (if `−N` is not specified), or the first unread message if there are no new messages, or the first message if there are no new or unread messages, followed by a prompt indicating `mailx` can accept regular commands (see 4.40.7.2); this is termed *command mode*. When `mailx` is invoked using the Send Mode synopsis and standard input is a terminal, if no subject is specified on the command line and the `asksub` variable is set, a prompt for the subject shall be written. At this point `mailx` is in *input mode*. This input mode is also entered when using one of the Receive Mode synopsis forms and a reply or new message is composed using the `reply`, `Reply`, or `mail` commands. When the message is typed and the end of message is encountered, the message shall be passed to the mail delivery software. Commands can be entered by beginning a line with the escape character [by default, tilde (~)] followed by a single command letter and optional arguments. See 4.40.7.3 for a summary of these commands.

NOTE: For notational convenience, this clause uses the default escape character, tilde, in all references and examples.

At any time, the behavior of `mailx` shall be governed by a set of environmental and internal variables. These are flags and valued parameters that can be set and cleared via the `set` and `unset` commands. See 4.40.5.3 and 4.40.7.1 for summaries of these parameters.

Regular commands are of the form:

[*command* **] [** *msglist* **] [** *argument* ... **]**

If no *command* is specified in command mode, `print` shall be assumed. In input mode, commands shall be recognized by the escape character, and lines not treated as commands shall be taken as input for the message.

In command mode, each message shall be assigned a sequential number, starting with 1.

All messages have a state that affects how they are displayed in the header summary and how they are retained or deleted upon termination of `mailx`. There is at any time the notion of a *current* message, marked by a > at the beginning of a line in the header summary. All messages are in one of the following states:

new The message is present in the system mailbox and has not been viewed by the user or moved to any other state. Messages in state *new* when `mailx` quits shall be retained in the system mailbox.

unread The message has been present in the system mailbox for more than one invocation of `mailx` and has not been viewed by the user or moved to any other state. Messages in state *unread* when `mailx` quits shall be retained in the system mailbox.

read The message has been processed by one of the following commands: `~f`, `~m`, `~F`, `~M`, `copy`, `mbox`, `next`, `pipe`, `print`, `Print`, `top`, `type`, `Type`, `undelete`. The `delete`, `dp`, and `dt` commands may also cause the next message to be marked as *read*, depending on the value of the `autoprint` variable. Messages that are in the system mailbox and in state *read* when `mailx` quits shall be
saved in the *mbox*, unless the internal variable `hold` was set. 7
Messages that are in the *mbox* or in a secondary mailbox and in 7
state *read* when `mailx` quits shall be retained in their current location.

deleted The message has been processed by one of the following com-
mands: `delete`, `dp`, `dt`. A message processed by `save` shall be 7
in state *deleted* unless the internal variable `keepsave` was set. 7
Messages in state *deleted* when `mailx` quits shall be deleted.

preserved The message has been processed by a `preserve` command. When `mailx` quits, the message shall be retained in its current location.

The header-summary line for each message shall indicate the state of the message.

Many commands take an optional list of messages (*msglist*) on which to operate, which defaults to the current message. A *msglist* is a list of message specifications separated by <blank>s, which can include:

n Message number *n*.

`+` The next undeleted message, or the next deleted message for the 7
`undelete` command. 7

`-` The next previous undeleted message, or the next previous 7
deleted message for the `undelete` command. 7

`.` The current message.

`^` The first undeleted message, or the first deleted message for the `undelete` command.

`$` The last message.

`*` All messages.

n−*m* An inclusive range of message numbers.

address All messages from *address*; any address as shown in a header summary shall be matchable in this form.

`/`*string* All messages with *string* in the subject line (case ignored).

`:`*c* All messages of type *c*, where *c* shall be one of:

- `d` Deleted messages
- `n` New messages
- `o` Old messages (any not in state *read* or *new*).
- `r` Read messages
- `u` Unread messages

Other commands take an optional message (*message*) on which to operate, which 7
defaults to the current message. All of the forms allowed for *msglist* are also 7
allowed for *message*, but if more than one message is specified, only the first shall 7
be operated on. 7

Other arguments are usually arbitrary strings whose usage depends on the command involved.

At startup time, `mailx` shall take the following steps in sequence:

(1) Establish all variables at their stated default values.

(2) Process command-line options, overriding corresponding default values.

(3) Import any of the **DEAD**, **EDITOR**, **MBOX**, **LISTER**, **PAGER**, **SHELL**, or **VISUAL** variables that are present in the environment, overriding the corresponding default values.

(4) Read `mailx` commands from an unspecified system startup file, unless 7
the `−n` option is given, to initialize any internal `mailx` variables and 7
aliases.

(5) Process the startup file of `mailx` commands named in the user **MAILRC** variable.

Most regular `mailx` commands are valid inside startup files, the most common use being to set up initial display options and alias lists. The following commands shall be invalid in the startup file: `!`, `edit`, `hold`, `mail`, `preserve`, `reply`, `Reply`, `shell`, and `visual`. Any errors in the startup file shall either cause `mailx` to terminate with a diagnostic message and a nonzero status, or to continue after writing a diagnostic message, ignoring the remainder of the lines in the startup file.

A blank line in a startup file shall be ignored. 7

4.40.7.1 `mailx` Internal Variables

The following variables are internal `mailx` variables. Each internal variable can be set via the `mailx set` command at any time. The `unset` and `set no`*name* commands can be used to erase variables.

In the following list, variables shown as

`variable`

represent Boolean values. Variables shown as

`variable=`*value*

shall be assigned string or numeric values. For string values, the rules in 4.40.7.2 concerning file names and quoting also apply.

The defaults specified here may be changed by the implementation-specific system startup file unless the user specifies the `−n` option.

`append`
: Append messages to the end of the *mbox* file upon termination instead of placing them at the beginning. The default shall be `noappend`. This variable shall not affect the `save` command when saving to the *mbox* .

`ask`
`asksub`
: Prompt for a subject line on outgoing mail if one is not specified on the command line with the `−s` option. The `ask` and `asksub` forms are synonyms; the system shall refer to `asksub` and `noasksub` in its messages, but shall accept `ask` and `noask` as user input to mean `asksub` and `noasksub`. It shall not be possible to set both `ask` and `noasksub`, or vice versa. The default shall be `asksub`, but no prompting shall be done if standard input is not a terminal.

`askbcc`
: Prompt for the Bcc list. The default shall be `noaskbcc`.

`askcc`
: Prompt for the Cc list. The default shall be `noaskcc`.

`autoprint`
: Enable automatic writing of messages after `delete` and `undelete` commands. The default shall be `noautoprint`.

`bang`
: Enable the special-case treatment of exclamation-marks (`!`) in escape command lines; see 4.40.7.2.41 and 4.40.7.3. The default shall be `nobang`, disabling the expansion of `!` in the *command* argument to the `~!` command and the `~<!`*command* escape.

`cmd=`*command*
: Set the default command to be invoked by the `pipe` command; see 4.40.7.2.21. The default shall be `nocmd`.

`crt=`*number*
: Pipe messages having more than *number* lines through the command specified by the value of the **PAGER** variable. The default shall be `nocrt`. If it is set to null, the value used is implementation defined.

`dot` When `dot` is set, a period on a line by itself during message input from a terminal shall also signify end-of-file (in addition to normal end-of-file). The default shall be `nodot`. If `ignoreeof` is set (see below), a setting of `nodot` shall be ignored and the period shall be the only method to terminate input mode.

`escape=`*c* Set the command escape character to be the character *c*. By default, the command escape character shall be tilde. If `escape` is unset, tilde shall be used; if it is set to null, command escaping shall be disabled.

`flipr` Reverse the meanings of the `R` and `r` commands; see 4.40.7.2.25 and 4.40.7.2.26. The default shall be `noflipr`.

`folder=`*directory*
The default directory for saving mail files. User-specified filenames beginning with a plus-sign (+) shall be expanded by preceding the filename with this directory name to obtain the real pathname. If *directory* does not start with a slash (/), the contents of **HOME** shall be prefixed to it. The default shall be `nofolder`. If `folder` is unset or set to null, user-specified filenames beginning with '`+`' shall refer to files in the current directory that begin with the literal `+` character. See also `outfolder` below. The `folder` value need not affect the processing of the files named in **MBOX** and **DEAD** .

`header` Enable writing of the header summary when entering `mailx` in Receive Mode. The default shall be `header`.

`hold` Preserve all messages that are read in the system mailbox instead of putting them in the *mbox* save file. The default shall be `nohold`.

`ignore` Ignore interrupts while entering messages. The default shall be `noignore`.

`ignoreeof` Ignore normal end-of-file during message input. Input can be terminated only by entering a period (`.`) on a line by itself or by the `~.` command escape. The default shall be `noignoreeof`. See also `dot` above.

`indentprefix=`*string*
A string that shall be prefixed to each line that is inserted into the message by the `~m` command escape. This variable shall default to one `<tab>` character.

`keep` When a system mailbox, secondary mailbox, or *mbox* is empty, truncate it to zero length instead of removing it. The default shall be `nokeep`.

`keepsave` Keep messages that have been saved in other files in the system mailbox instead of deleting them. The default shall be `nokeepsave`.

`metoo` Suppress the deletion of the login name of the user from the recipient list when replying to a message or sending to a group. The default shall be `nometoo`.

`outfolder` Cause the files used to record outgoing messages to be located in the directory specified by the `folder` variable unless the pathname is absolute. The default shall be `nooutfolder`. See the `record` variable.

`page` Insert a `<form-feed>` after each message sent through the pipe created by the `pipe` command. The default shall be `nopage`.

`prompt=`*string*

Set the command-mode prompt to *string*. If *string* is null or if `noprompt` is set, no prompting shall occur. The default shall be to prompt with the string `"? "`.

`quiet` Refrain from writing the opening message and version when entering `mailx`. The default shall be `noquiet`.

`record=`*file*

Record all outgoing mail in the file with the pathname *file*. The default shall be `norecord`. See also `outfolder` above.

`save` Enable saving of messages in the dead-letter file on interrupt or delivery error. See the variable **DEAD** for the location of the dead-letter file. The default shall be `save`.

`screen=`*number*

Set the number of lines in a screenful of headers for the `headers` and `z` commands. If `screen` is not specified, a value based on the terminal type identified by the **TERM** environment variable, the window size, and/or the baud rate shall be used.

`showto` When the sender of the message was the user who is invoking `mailx`, write the information from the `"To:"` line instead of the `"From:"` line in the header summary. The default shall be `noshowto`.

`sign=`*string*

Set the variable inserted into the text of a message when the `~a` command escape is given. The default shall be `nosign`. The character sequences `\t` and `\n` shall be recognized in the variable as `<tab>` and `<newline>`, respectively. (See also `~i` in 4.40.7.3).

`Sign=`*string*

Set the variable inserted into the text of a message when the `~A` command escape is given. The default shall be `noSign`. The character sequences `\t` and `\n` shall be recognized in the variable as `<tab>` and `<newline>`, respectively. (See also `~i` in 4.40.7.3).

`toplines=`*number*

Set the number of lines of the message to write with the `top` command. The default shall be 5.

4.40.7.2 `mailx` Commands

The following `mailx` commands shall be implemented. In the following list,
''header'' refers to lines from the message header, as shown in 4.40.6.3. ''Header- 7
line'' refers to lines within the header that begin with one or more non-white- 7
space characters, immediately followed by a colon and white space and continuing 7
until the next line beginning with a non-white-space character or an empty line. 7
''Header-field'' refers to the portion of a header line prior to the first colon in that line.

For each of the commands listed below, the command can be entered as the abbreviation (those characters in the Synopsis command word preceding the `[`), the full command (all characters shown for the command word, omitting the `[` and `]`), or any truncation of the full command down to the abbreviation. For example, the `exit` command (shown as `ex[it]` in the Synopsis) can be entered as `ex`, `exi`, or `exit`.

The arguments to commands can be quoted, using the following methods:

— An argument can be enclosed between paired double-quotes (`" "`) or single-quotes (`' '`); any white space, shell word expansion, or backslash characters within the quotes shall be treated literally as part of the argument. A double-quote shall be treated literally within single-quotes and vice versa. These special properties of the quote marks shall occur only when they are paired at the beginning and end of the argument.

— A backslash outside of the enclosing quotes shall be discarded and the following character treated literally as part of the argument.

— An unquoted backslash at the end of a command line shall be discarded and the next line shall continue the command.

File names, where expected, shall be subjected to the process of shell word expansions (see 3.6); if more than a single pathname results and the command is
expecting one file, the effects are unspecified. If the file name begins with an
unquoted plus-sign, it shall not be expanded, but treated as the named file (less 7
the leading plus) in the `folder` directory. (See the `folder` variable.) 7

4.40.7.2.1 Declare aliases

Synopsis : `a[lias]` [*alias* [*address* ...]] 7
Synopsis : `g[roup]` [*alias* [*address* ...]] 7

Add the given addresses to the alias specified by *alias* . The names shall be substi- 7
tuted when *alias* is used as a recipient address specified by the user in an outgoing message (i.e., other recipients addressed indirectly through the `reply` command shall not be substituted in this manner). Mail address alias substitution shall apply only when the alias string is used as a full address; e.g., when `hlj` is an alias, `hlj@posix.com` does not trigger the alias substitution. If no arguments are given, write a listing of the current aliases to standard output. If only an *alias* argument is given, write a listing of the specified alias to standard output. These listings need not reflect the same order of addresses that were entered.

4.40.7.2.2 Declare alternates

Synopsis : `alt[ernates]` *name* ...

Declare a list of alternate names for the user's login. When responding to a message, these names shall be removed from the list of recipients for the response. The comparison of names shall be in a case-insensitive manner. With no arguments, `alternates` shall write the current list of alternate names.

4.40.7.2.3 Change current directory

Synopsis : `cd` [*directory*]
Synopsis : `ch[dir]` [*directory*]

Change directory. If *directory* is not specified, the contents of **HOME** shall be used.

4.40.7.2.4 Copy messages

Synopsis : `c[opy]` [*file*]
Synopsis : `c[opy]` [*msglist*] *file*

Copy messages to the file named by the pathname *file* without marking the mes-
sages as saved. Otherwise, it shall be equivalent to the `save` command. 7

4.40.7.2.5 Delete messages

Synopsis : `d[elete]` [*msglist*]

Mark messages for deletion from the mailbox. The deletions shall not occur until
`mailx` quits (see 4.40.7.2.24) or changes mailboxes (see 4.40.7.2.10). If `auto-
print` is set, the next message after the last one deleted shall be written (see 8
4.40.7.1); if there is no subsequent message, the previous message, if it exists, 8
shall be written. In the case of no subsequent or previous message, or when 7
`noautoprint` is set, the `mailx` prompt shall be written. 7

4.40.7.2.6 Discard header fields

Synopsis : `di[scard]` [*header-field* ...]
Synopsis : `ig[nore]` [*header-field* ...] 7

Suppress the specified header fields when writing messages. Specified *header-* 7
fields shall be added to the list of suppressed header fields. Examples of header 7
fields to ignore are `status` and `cc`. The fields shall be included when the mes-
sage is saved. The `Print` and `Type` commands shall override this command. The
comparison of header fields shall be in a case-insensitive manner. If no argu- 7
ments are specified, write a list of the currently suppressed header fields to stan- 7
dard output; the listing need not reflect the same order of header fields that were 7
entered. 7

If both `retain` (see 4.40.7.2.27) and `discard` commands are given, `discard` 7
commands shall be ignored. 7

4.40.7.2.7 Delete messages and display

Synopsis : `dp` [*msglist*]
Synopsis : `dt` [*msglist*]

Delete the specified messages from the mailbox and write the next message after
the last one deleted. If there is no subsequent message, the `mailx` prompt shall 7
be written. 7

4.40.7.2.8 Edit messages

Synopsis : `e[dit]` [*msglist*]

Edit the given messages. The messages shall be placed in a temporary file and
the utility named by the **EDITOR** variable (see 4.40.5.3) shall be invoked to edit 7
the file. The default editor is unspecified. 7

4.40.7.2.9 Exit

Synopsis : `ex[it]`
Synopsis : `x[it]`

Exit from `mailx` without changing the mailbox. No messages shall be saved in the *mbox* (see also `quit`).

4.40.7.2.10 Change folder

Synopsis : `fi[le]` [*file*]
Synopsis : `fold[er]` [*file*]

Quit (see 4.40.7.2.24) from the current file of messages and read in the file named by the pathname *file* . If no argument is given, the name and status of the current mailbox shall be written.

Several unquoted special characters shall be recognized when used as *file* names, with the following substitutions:

%	The system mailbox for the invoking user. 7
%*user*	The system mailbox for *user* .
#	The previous file.
&	The current *mbox* .
+*file*	The named file in the `folder` directory. (See the `folder` variable.)

The default file shall be the current mailbox.

4.40.7.2.11 Display list of folders

Synopsis : `folders`

Write the names of the files in the directory set by the `folder` variable (see 4.40.5.3).

4.40.7.2.12 Display header summary for specified messages

Synopsis : `f[rom]` [*msglist*]

Write the header summary for the specified messages.

4.40.7.2.13 Display header summary

Synopsis : `h[eaders]` [*message*]

Write the page of headers that includes the message specified. The `screen` variable sets the number of headers per page. See also the `z` command.

4.40.7.2.14 Help

Synopsis : `hel[p]`
Synopsis : `?` 7

Write a summary of commands.

4.40.7.2.15 Hold messages

Synopsis : `ho[ld]` [*msglist*]
Synopsis : `pre[serve]` [*msglist*]

Mark the messages in *msglist* to be retained in the mailbox when `mailx` terminates. This shall override any commands that might previously have marked the messages to be deleted. During the current invocation of `mailx`, only the `delete`, `dp`, or `dt` commands shall remove the *preserve* marking of a message.

4.40.7.2.16 Execute commands conditionally

Synopsis : `i[f] s|r`
mail-command s
`el[se]`
mail-command s
`en[dif]`

Execute commands conditionally, where `if s` shall execute the following *mail-command* s, up to an `else` or `endif`, if the program is in Send Mode, and `if r` shall cause the *mail-command* s to be executed only in Receive Mode.

4.40.7.2.17 List available commands

Synopsis : `l[ist]`

Write a list of all commands available. No explanation shall be given.

4.40.7.2.18 Mail a message

Synopsis : `m[ail]` *address* . . .

Mail a message to the specified addresses or aliases.

4.40.7.2.19 Direct messages to mbox

Synopsis : `mb[ox]` [*msglist*]

Arrange for the given messages to end up in the *mbox* save file when `mailx` terminates normally. (See **MBOX** in 4.40.5.3 for a description of this file.) See also the `exit` and `quit` commands.

4.40.7.2.20 Process next specified message

Synopsis : `n[ext]` [*message*]

8

Go to the next message matching *message* . 8

4.40.7.2.21 Pipe message

Synopsis : `pi[pe]` [[*msglist*] *command*] 7
Synopsis : `|` [[*msglist*] *command*] 7

Pipe the messages through the given *command* by invoking the command interpreter specified by **SHELL** (see 4.40.5.3) with two arguments: `−c` and *command* . (See also `sh −c` in 4.56.) The command shall be given as a single argument. Quoting, described previously, can be used to accomplish this. If no arguments are given, the current message shall be piped through the command specified by the value of the `cmd` variable. If the `page` variable is set, a `<form-feed>` character shall be inserted after each message. 7 7

4.40.7.2.22 Display message with headers

Synopsis : `P[rint]` [*msglist*]
Synopsis : `T[ype]` [*msglist*]

Write the specified messages, including all header lines, to standard output. Override suppression of lines by the `discard`, `ignore`, and `retain` commands. If `crt` is set, the messages longer than the number of lines specified by the `crt` variable shall be paged through the command specified by the **PAGER** environment variable. (See 4.40.5.3).

4.40.7.2.23 Display message

Synopsis : `p[rint]` [*msglist*]
Synopsis : `t[ype]` [*msglist*]

Write the specified messages to standard output. If `crt` is set, the messages longer than the number of lines specified by the `crt` variable shall be paged through the command specified by the **PAGER** environment variable. (See 4.40.5.3).

4.40.7.2.24 Quit

Synopsis : `q[uit]`
Synopsis : *end-of-file*

Terminate `mailx`, storing messages that were read in *mbox* (if the current mailbox is the system mailbox and unless `hold` is set), deleting messages that have been explicitly saved (unless `keepsave` is set), discarding messages that have been deleted, and saving all remaining messages in the mailbox.

4.40.7.2.25 Reply to a message list

Synopsis : `R[eply]` [*msglist*]
Synopsis : `R[espond]` [*msglist*]

Mail a reply message to the sender of each message in the *msglist* . The subject 7
line shall be formed by concatenating `"Re: "` (unless it already begins with that
string) and the subject from the first message. If `record` is set to a file name, the
response shall be saved at the end of that file.

See also the `flipr` variable in 4.40.7.1. 7

4.40.7.2.26 Reply to a message

Synopsis : `r[eply]` [*message*]
Synopsis : `r[espond]` [*message*]

Mail a reply message to all recipients included in the header of the message. The
subject line shall be formed by concatenating `"Re: "` (unless it already begins
with that string) and the subject from the message. If `record` is set to a file 7
name, the response shall be saved at the end of that file.

See also the `flipr` variable in 4.40.7.1. 7

4.40.7.2.27 Retain header fields

Synopsis : `ret[ain]` [*header-field* ...]

Retain the specified header fields when writing messages. This command shall 7
override all `discard` and `ignore` commands. The comparison of header fields 7
shall be in a case-insensitive manner.

4.40.7.2.28 Save messages

Synopsis : `s[ave]` [*file*]
Synopsis : `s[ave]` [*msglist*] *file*

Save the specified messages in the file named by the pathname *file*, or the *mbox* if the *file* argument is omitted. The file shall be created if it does not exist; otherwise, the messages shall be appended to the file. The message shall be deleted from the mailbox when `mailx` terminates unless `keepsave` is set.

4.40.7.2.29 Set variables

Synopsis : `se[t]` [*name* [=[*string*]] . . .] [*name* =*number* . . .] [`no`*name* . . .] 7

Define one or more variables called *name* . The variable can be given a null, string, 7
or numeric value. Quoting and backslash escapes can occur anywhere in *string* ,
as described previously, as if the *string* portion of the argument were the entire
argument. The forms *name* and *name*= shall be equivalent to *name* =`""` for vari- 7
ables that take string values. The `set` command without arguments shall write a 7
list of all defined variables and their values. The `no`*name* form shall be
equivalent to ‘`unset` *name* ’’.

4.40.7.2.30 Invoke a shell

Synopsis : `sh[ell]`

Invoke an interactive command interpreter (see also **SHELL** in 4.40.5.3).

4.40.7.2.31 Display message size

Synopsis : `si[ze]` [*msglist*]

Write the size in bytes of each of the specified messages. 7

4.40.7.2.32 Read `mailx` commands from a file

Synopsis : `so[urce]` *file*

Read and execute commands from the file named by the pathname *file* and return 7
to command mode.

4.40.7.2.33 Display beginning of messages

Synopsis : `to[p]` [*msglist*]

Write the top few lines of each of the specified messages. If the `toplines` vari- 7
able is set, it is taken as the number of lines to write. The default shall be 5.

4.40.7.2.34 Touch messages

Synopsis : `tou[ch]` [*msglist*]

Touch the specified messages. If any message in *msglist* is not specifically deleted 7
nor saved in a file, it shall be placed in the *mbox* upon normal termination. See 7
`exit` and `quit`.

4.40.7.2.35 Delete aliases

Synopsis : `una[lias]` [*alias*] . . .

Delete the specified alias names. If a specified alias does not exist, the results are unspecified.

4.40.7.2.36 Undelete messages

Synopsis : `u[ndelete]` [*msglist*]

Remove the deleted markings from the specified messages. If `autoprint` is set,
the last message of those restored shall be written. If *msglist* is not specified, it 7
shall default to the first deleted message following the current message that has 7
not been undeleted if there is one, or the last deleted message preceding the 7
current message that has not been undeleted otherwise. 7

4.40.7.2.37 Unset variables

Synopsis : `uns[et]` *name* . . .

Cause the specified variables to be erased.

4.40.7.2.38 Edit message with full-screen editor

Synopsis : `v[isual]` [*msglist*]

Edit the given messages with a screen editor. The messages shall be placed in a
temporary file, and the utility named by the **VISUAL** variable (see 4.40.5.3) shall 7
be invoked to edit the file. The default editor shall be `vi`. 7

4.40.7.2.39 Write messages to a file

Synopsis : `w[rite]` [*msglist*] *file*

Write the given messages to the file specified by the pathname *file*, minus the 7
message header. Otherwise, it shall be equivalent to the `save` command.

4.40.7.2.40 Scroll header display

Synopsis : `z[+|–]`

Scroll the header display forward (if + is specified or if no option is specified) or 7
backward (if - is specified) one screenful. The number of headers written shall be 7
set by the `screen` variable. 7

4.40.7.2.41 Invoke shell command

Synopsis : `!`*command*

Invoke the command interpreter specified by **SHELL** (see 4.40.5.3) with two argu- 7
ments: `–c` and *command* . (See also `sh –c` in 4.56.) If the `bang` variable is set, 7
each unescaped occurrence of ! in command shall be replaced with the command 7
executed by the previous ! command or ~! command escape. 7

4.40.7.2.42 Null command

Synopsis : # *comment*

This null command (comment) shall be ignored by `mailx`.

4.40.7.2.43 Display current message number

Synopsis : `=`

Write the current message number.

7

4.40.7.3 **`mailx`** Command Escapes

The following commands can be entered only from input mode, by beginning a line with the escape character [by default, tilde (~)]. See the `escape` variable description for changing this special character. The format for the commands is:

<ESC ><command-char><separator> [*<argument(s)>*]

where the *<separator>* can be zero or more `<blank>`s.

In the following descriptions, the argument *command* (but not *mailx-command)*
shall be a shell command string. Any string acceptable to the command inter- 7
preter specified by the **SHELL** variable (see 4.40.5.3) when it is invoked as `-c` 7
command_string (see 4.56 for a description of `sh -c`) shall be valid. The command 7
can be presented as multiple arguments (i.e., quoting is not required).

Command escapes that are listed with *msglist* or *mailx-command* arguments are invalid in Send Mode and produce unspecified results.

~! *command* — Invoke the command interpreter specified by **SHELL** (see 7
4.40.5.3) with two arguments: `-c` and *command* ; and then 7
return to input mode. If the `bang` variable is set, each unes- 7
caped occurrence of ! in *command* shall be replaced with the 7
command executed by the previous ! command or ~! com- 7
mand escape. 7

`~.` — Simulate end-of-file (terminate message input).

`~:` *mailx-command*
`~_` *mailx-command*
— Perform the command-level request.

`~?` — Write a summary of command escapes.

`~A` — This shall be equivalent to `~i Sign`.

`~a` — This shall be equivalent to `~i sign`.

`~b` *name* ... — Add the *name* s to the blind carbon copy (Bcc) list.

`~c` *name* ... — Add the *name* s to the carbon copy (Cc) list.

`~d` — Read in the dead-letter file. See **DEAD** (4.40.5.3) for a description of this file.

`~e` — Invoke the editor, as specified by the **EDITOR** environment variable (see 4.40.5.3), on the partial message.

`~f` [*msglist*] Forward the specified messages. The specified messages shall be inserted into the current message without alteration. This command escape shall also insert message headers into the message with field selection affected by the `discard`, `ignore`, and `retain` commands.

`~F` [*msglist*] This shall be the equivalent of the `~f` command escape, except that all headers shall be included in the message, regardless of previous `discard`, `ignore`, and `retain` commands.

`~h` If standard input is a terminal, prompt for a Subject line and the To, Cc, and Bcc lists. Other implementation-defined headers may also be presented for editing. If the field is written with an initial value, it can be edited as if it had just been typed.

`~i` *string* Insert the value of the named variable, followed by a `<newline>`, into the text of the message. If the string is unset or null, the message shall not be changed. 7

`~m` [*msglist*] Insert the specified messages into the message, prefixing nonempty lines with the string in the `indentprefix` variable. This command escape shall also insert message headers into the message, with field selection affected by the `discard`, `ignore`, and `retain` commands.

`~M` [*msglist*] This shall be the equivalent of the `~m` command escape, except that all headers shall be included in the message, regardless of previous `discard`, `ignore`, and `retain` commands.

`~p` Write the message being entered. If the message is longer 7
than `crt` lines (see 4.40.7.1), the output shall be paginated 7
as described for the **PAGER** variable (see 4.40.5.3). 7

`~q` Quit (see 4.40.7.2.24) from input mode by simulating an interrupt; see 4.40.5.4. If the body of the message is not empty, the partial message shall be saved in the dead-letter file. See **DEAD** (4.40.5.3) for a description of this file.

`~r` *file*
`~<` *file*
`~<` `!`*command* Read in the file specified by the pathname *file*. If the argument begins with an exclamation-mark (`!`), the rest of the
string shall be taken as an arbitrary system command; the 7
command interpreter specified by **SHELL** (see 4.40.5.3) shall 7
be invoked with two arguments: `−c` and *command* . (See also 7
`sh −c` in 4.56.) The standard output of *command* shall be 7
inserted into the message. 7

`~s` *string* Set the subject line to *string* .

`~t` *name* ... Add the given *name* s to the To list.

`~v` Invoke the full-screen editor, as specified by the **VISUAL** environment variable (see 4.40.5.3), on the partial message.

`~w` *file* Write the partial message, without the header, onto the file named by the pathname *file*. The file shall be created or the message shall be appended to it if the file exists.

`~x` Exit as with `~q`, except the message shall not be saved in the dead-letter file.

`~|` *command*
Pipe the body of the message through the given *command* by 7
invoking the command interpreter specified by **SHELL** (see 7
4.40.5.3) with two arguments: `-c` and *command*. If the *com-* 7
mand returns a successful exit status, the standard output 7
of the command shall replace the message. Otherwise the message shall remain unchanged. If the *command* fails, an error message giving the exit status shall be written.

4.40.8 Exit Status

When the `-e` option is specified, the `mailx` utility shall exit with one of the following values:

0 Mail was found.

>0 Mail was not found or an error occurred.

Otherwise, `mailx` utility shall exit with one of the following values:

0 Successful completion.

>0 An error occurred.

4.40.9 Consequences of Errors

When in input mode (Receive Mode) or Send Mode: 7

— If an error is encountered processing a command escape (see 4.40.7.3), a 7
diagnostic message shall be written to standard error, and the message 7
being composed may be modified, but this condition shall not prevent the 7
message from being sent. 7

— Other errors shall prevent the sending of the message. 7

When in command mode:

— Default.

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

Usage, Examples

The format of the system mailbox is intentionally unspecified. Not all systems will implement system mailboxes as flat files, particularly with the advent of multimedia mail messages. Some system mailboxes may be multiple files, others records in a database. The internal format of the messages themselves are specified with the historical format from Version 7, but only after they have been saved in some file other than the system mailbox. This was done so that many historical applications expecting text-file mailboxes will not be broken.

Some new formats for messages can be expected in the future, probably including binary data, bit maps, and various multimedia objects. As described here, `mailx` is not prohibited from handling such messages, but it must store them as text files in secondary mailboxes (unless some extension, such as a variable or command-line option, is used to change the stored format). Its method of doing so is implementation defined and might include translating the data into text-file-compatible or readable form or omitting certain portions of the message from the stored output.

Some historical implementations support the following internal variable:

`debug` Enable verbose diagnostics for debugging. Messages shall not be delivered. The default shall be `nodebug`.

The `edit` and `visual` commands merely edit the specified messages in a temporary file. They do not modify the contents of those messages in the mailbox; such a capability could be added as an extension, such as by using different command names.

The `discard`/`ignore` and `retain` commands are not inverses of each other. The `retain` command discards all header-fields except those explicitly retained. The `discard` command keeps all header-fields except those explicitly discarded. After a `retain` command is specified, `discard` and `ignore` commands are ignored.

The restriction on a subject line being {LINE_MAX} – 10 bytes is based on the historical format that consumes 10 bytes for `"Subject: "` and the trailing `<newline>`. Many historical mailers that a message may encounter on other systems will not be able to handle lines that long, however.

History of Decisions Made

Any references to networking have been removed as being outside the scope of the document.

The developers of the standard felt strongly that a method for applications to send messages to specific users was necessary. The obvious example is a batch utility, running noninteractively, that wishes to communicate errors or results to a user. However, the actual format, delivery mechanism, and method of reading the message are clearly beyond the scope of this standard.

The intent of this command is to provide a simple, portable interface for sending messages noninteractively. It merely defines a ''front-end'' to the historical mail system. It is suggested that implementations explicitly denote the sender and recipient in the body of the delivered message. Further specification of formats for either the message envelope or the message itself were deliberately not made, as the industry is in the midst of changing from the current standards to a more internationalized standard and it is probably incorrect, at this time, to require either one.

Implementations are encouraged to conform to the various delivery mechanisms desribed in the CCITT X.400 standards or to the equivalent Internet Standards described in Internet ''Request for Comment'' (RFC) documents RFC 819, RFC 822, RFC 920, RFC 921, and RFC 1123. Copies of these Internet RFCs can be obtained from the DDN Network Information Center, SRI International, Menlo Park, CA, USA 94025. *Editor's Note: This paragraph will be reworked editorially to use a bibliographic entry after the two standards are merged. Doing so now is rather a headache.*

The standard does not place any restrictions on the length of messages handled by `mailx`, and for delivery of local messages the only limitations should be the normal problems of available disk space for the target mail file. When sending messages to external machines, applications are advised to limit messages to less than 50 kilobytes because many mail gateways impose message-length restrictions. (Note that this is usually an administrative issue based on the amount of mail traffic and disk space available on the gateways. Therefore, there is no way for this standard to require implementations to guarantee delivery of long messages to remote systems.)

Like the utilities `logger` and `lp`, `mailx` is admittedly difficult to test. This was not deemed sufficient justification to exclude these utilities from the standard. It is also arguable that they are, in fact, testable, but that the tests themselves are not portable.

When `mailx` is being used by an application that wishes to receive the results as if none of the User Portability Utilities Option features were supported, the **DEAD** environment variable must be set to `/dev/null`. Otherwise, it may be subject to the file creations described in 4.40.5.4. Similarly, if the **MAILRC** environment variable is not set to `/dev/null`, historical versions of `mailx` and `Mail` read initialization commands from a file before processing begins. Since the initialization that a user specifies could alter the contents of messages an application is trying to send, such applications must set **MAILRC** to `/dev/null`.

The description of **LC_TIME** uses ''may affect'' because many historical implementations do not or cannot manipulate the date and time strings in the incoming mail headers. Some headers found in incoming mail do not have enough information to determine the time zone in which the mail originated, and, therefore, `mailx` cannot convert the date and time strings into the internal form that can then be parsed by routines like *strftime* () that can take **LC_TIME** settings into account. Changing all these times to a user-specified format is allowed, but not required. 7

The paginator selected when **PAGER** is null or unset is partially unspecified to 7
allow System V historical practice of using `pg` as the default. Bypassing the pagi- 7
nation function, such as by declaring that `cat` is the paginator, would not meet 7
with the intended meaning of this description. However, any ''portable user'' 7
would have to set **PAGER** explicitly to get his or her preferred paginator on all 7
systems. The paginator choice was made partially unspecified, unlike the 7
VISUAL editor choice (mandated to be `vi`) because most historical pagers follow a 7
common theme of user input, whereas editors differ dramatically. 7

Options to specify addresses as ''cc''(carbon-copy) or ''bcc''(blind-carbon-copy) 1
were considered to be format details and were omitted. 1

A zero exit status implies that all messages were *sent*, but it gives no assurances that any of them were actually *delivered*. The reliability of the delivery mechanism is unspecified and is an appropriate marketing distinction between systems.

In order to conform to the utility syntax guidelines, a solution was required to the optional *file* option-argument to `-f`. By making *file* an operand, the guidelines are satisfied and users remain portable. However, it does force implementations to support usage such as:

```
mailx -fin mymail.box
```

The `-F` was incorrectly stated in previous drafts to apply to Receive Mode, but it has only been in `mailx` Send Mode historically. It, along with the `Copy`, `Save`, `followup`, and `Followup` commands, has been omitted. Historical systems used these to save messages in a file whose name is ''derived from the author (or recipient) of the message.'' Because of some unusual network mail addresses currently in use, and because POSIX does not specify anything about the content of the mail headers, it is not possible for this standard to identify the algorithm used for this derivation; since files would therefore be created in an unpredictable manner, this is unacceptable for portable usage.

The `no`*name* method of unsetting variables is not present in all historical systems, but it is in System V and provides a logical set of commands corresponding to the format of the display of options from the `mailx set` command without arguments.

The `ask` and `asksub` variables are the names selected by BSD and System V, respectively, for the same feature. They are synonyms here.

The `mailx echo` command was not documented in the BSD version and has been omitted here because it is not obviously useful for interactive users.

The default prompt on the System V `mailx` is a question mark, on BSD `Mail` an 7
ampersand. Since POSIX.2 chose the `mailx` name, it kept the System V default, 7
assuming that BSD users would not have difficulty with this minor incompatibil- 7
ity (that they can override anyway). 7

The meanings of `r` and `R` are reversed between System V `mailx` and Sun OS `Mail`. 7
Once again, since POSIX.2 chose the `mailx` name, it kept the System V default, 7
but allows the Sun OS user to achieve the desired results using `flipr`, an internal 7

variable in System V `mailx`, although it has not been documented in the *SVID*. 7

The `indentprefix` variable, the `retain` and `unalias` commands, and the `~F` 7
and `~M` command escapes were adopted from 4.3BSD `Mail`. 7

The `version` command was not included because no sufficiently general specification of the version information could be devised that would still be useful to a portable user. This command name should be used by suppliers who wish to provide version information about the `mailx` command.

The ''implementation-specific (unspecified) system startup file'' has been tradition- 7
ally named `/etc/mailx.rc`, but this specific name and location are not required.

4.52 `read` — Read a line from standard input

⇒ **4.52.2 `read` Description.** *In subclause 4.52, add the following to the end of the second paragraph in the Description subclause:*

If standard input is a terminal device and the invoking shell is interactive, `read` shall prompt for a continuation line when:

— The shell reads an input line ending with a backslash, unless the `−r` option is specified.

— A here-document is not terminated after a `<newline>` is entered.

⇒ **4.52.5.3 `read` Environment Variables.** *In subclause 4.52, add the following environment variable in the correct sorted order to the Environment Variables subclause.*

PS 2 This variable shall contain the prompt string that an interactive shell shall write to standard error when a line ending with a backslash is read and the `−r` option was not specified, or if a here-document is not terminated after a `<newline>` is entered.

⇒ **4.51.6.2 `read` Standard Error.** *In subclause 4.52, replace the Standard Error subclause with the following.*

Used for diagnostic messages and prompts for continued input.

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

None.

4.56 `sh` — Shell, the standard command language interpreter

⇒ **4.56.1 `sh` Synopsis.** *Add the* `−b`*,* `−m`*, and* `−o` *option options to each of the three Synopsis lines with appropriate editorial wording indicating that these* 7
options need to be supported only when the UPUO is supported: 7

⇒ **4.56.3 `sh` Options.** *Change the first sentence of the second paragraph, beginning* 'The −a, −C, . . . '*'to:*

The `−a`, `−b`, `−C`, `−e`, `−f`, `−m`, `−n`, `−o` *option* , `−u`, `−v`, and `−x` options are described as part of the `set` utility in 3.14.11.

⇒ **4.56.3 `sh` Options.** *Remove the following sentence and NOTE:* 8

The behavior of an interactive shell is not fully specified by this standard. 8

NOTE: The preceding sentence is expected to change following the eventual approval of the UPE 8
supplement. 8

⇒ **4.56.5.3 `sh` Environment Variables.** *Modify the contents of subclause 4.56.5.3, Environment Variables, to add the following environment variables in the correct sorted order.*

FCEDIT 8
HISTFILE 8
HISTSIZE See the description of the `fc` environment variables in 8
5.12.5.3. This standard specifies the effects of these 8
variables only for systems supporting the User Porta- 8
bility Utilities Option. 8

MAIL This variable shall be interpreted as a pathname of the user's mailbox file for purposes of incoming mail notification. If this variable is set, the shell shall inform the user if the file named by the variable is created or if its modification time has changed. Informing the user shall be accomplished by writing a string of unspecified format to the standard error prior to the writing of the next primary prompt string after the completion of an interval defined by the **MAIL-CHECK** variable. The user shall be informed only if **MAIL** is set and **MAILPATH** is not set. This standard 7 specifies the effects of this variable only for systems supporting the User Portability Utilities Option.

MAILCHECK This variable shall be interpreted as a decimal integer value that specifies how often (in seconds) the shell shall check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** variables. The default value shall be 600 seconds. If set to zero, the shell shall check before issuing each primary prompt. This standard specifies the effects of this variable only for systems supporting the User Portability Utilities Option.

MAILPATH This variable shall be interpreted as a list of pathnames and optional messages separated by colons. If this variable is set, the shell shall inform the user if any of the files named by the variable are created or if any of their modification times change. (See the preceding entry for **MAIL** for descriptions of mail arrival and user informing.) Each pathname can be followed by `%` and a string that shall be subjected to parameter expansion and written to standard error when the modification time changes. If a `%` character in the pathname is preceded by a backslash, it shall be treated as a literal `%` in the pathname. The default message is unspecified. This standard specifies the effects of this variable only for systems supporting the User Portability Utilities Option.

The **MAILPATH** environment variable shall take precedence over the **MAIL** variable.

⇒ **4.56.7 `sh` Extended Description.** *Replace the subclause 4.56.7, Extended Description, with the following.*

4.56.7 Extended Description

See the description of the shell command language in Section 3.

On systems supporting the User Portability Utilities Option, the following additional capabilities shall be supported.

4.56.7.1 Command History List

When the `sh` utility is being used interactively, it shall maintain a list of commands previously entered from the terminal in the file named by the **HISTFILE** environment variable. The type, size, and internal format of this file are unspecified. Multiple `sh` processes can share access to the file for a user, if file access permissions allow this; see the description of the **HISTFILE** environment variable in 5.12.5.3.

4.56.7.2 Command Line Editing

When `sh` is being used interactively from a terminal, the current command and the command history (see `fc` in 5.12) can be edited using *vi-mode* command-line editing. This mode uses commands, described below, similar to a subset of those described in the `vi` (see 5.35) utility. Implementations may offer other command-line editing modes corresponding to other editing utilities.

The command `set -o vi` shall enable vi-mode editing and place `sh` into `vi` insert mode (see 4.56.7.3). This command shall also disable any other editing mode that the implementation may provide. The command `set +o vi` shall disable vi-mode editing.

Certain block-mode terminals may be unable to support shell command line editing. If a terminal is unable to provide either edit mode, it need not be possible to `set -o vi` when using the shell on this terminal.

In the following subclauses, the characters *erase*, *interrupt*, *kill*, and *end-of-file* are those set by the `stty` utility; see 4.59.

4.56.7.3 Vi-Mode Command Line Editing

With vi-mode enabled, `sh` can be switched between insert mode and command mode.

When in insert mode, an entered character shall be inserted into the command line, except as noted in 4.56.7.4. Upon entering `sh` and after termination of the previous command, `sh` shall be in insert mode.

Typing `<ESC>` shall switch `sh` into command mode (see 4.56.7.5). In command mode, an entered character shall either invoke a defined operation, be used as part of a multicharacter operation, or be treated as an error. A character that is not recognized as part of an editing command shall terminate any specific editing command and shall alert the terminal. Typing the *interrupt* character in command mode shall cause `sh` to terminate command-line editing on the current command line, re-issue the prompt on the next line of the terminal, and reset the command history (see 5.12) so that the most recently executed command is the ‘‘previous’’ command (i.e., the command that was being edited when it was interrupted is not re-entered into the history).

In the following subclauses, the phrase ‘‘move the cursor to the beginning of the word’’ shall mean ‘‘move the cursor to the first character of the current word’’ and the phrase ‘‘move the cursor to the end of the word’’ shall mean ‘‘move the cursor to the last character of the current word.’’ The phrase ‘‘beginning of the command line’’ indicates the point between the end of the prompt string issued by the shell (or the beginning of the terminal line, if there is no prompt string) and the first character of the command text.

4.56.7.4 Vi Line Editing Insert Mode

While in insert mode, any character typed shall be inserted in the current command line, unless it is from the following set.

`<newline>` Execute the current command line being edited.

erase Delete the character previous to the current cursor position and move the current cursor position back one character. In insert mode, characters shall be erased from both the screen and the buffer when backspacing.

interrupt Terminate command-line editing with the same effects as described for interrupting command mode; see 4.56.7.3.

kill Clear all the characters from the input line.

`control-V` Insert the next character input, even if the character is otherwise a special insert mode character.

`control-W` Delete the characters from the one preceding the cursor to the preceding word boundary. The word boundary in this case is the closer to the cursor of either the beginning of the line or a character that is in neither the `blank` nor `punct` character classification of the current locale.

`\` If followed by the *erase* or *kill* character, that character shall be inserted into the input line. Otherwise, the backslash itself shall be inserted into the input line.

end-of-file Interpreted as the end of input in `sh`. This interpretation shall occur only at the beginning of an input line. If end-of-file is entered other than at the beginning of the line, the results are unspecified.

`<ESC>` Place `sh` into command mode.

4.56.7.5 Vi Line Editing Command Mode

In command mode for the command-line editing feature, decimal digits not beginning with 0 that precede a command letter shall be remembered. Some commands use these decimal digits as a count number that affects the operation.

The term *motion command* in the following list represents one of the commands:

`<space>`	`0`	`b`	`F`	`l`	`W`	`^`
`$`	`;`	`E`	`f`	`T`	`w`	`|`
`,`	`B`	`e`	`h`	`t`		

Any command that modifies the current line shall cause a copy of the current line to be made at the end of the command history, the current line shall become that copy, and the edit shall be performed on that copy.

Any command that is preceded by *count* shall take a count (the numeric value of any preceding decimal digits). Unless otherwise noted, this count shall cause the specified operation to repeat by the number of times specified by the count. Also unless otherwise noted, a *count* that is out of range is considered an error condition and shall alert the terminal, but neither the cursor position, nor the command line, shall change.

The terms *word* and *bigword* are used as defined in the `vi` description; see 5.35.7.1. The term *save buffer* corresponds to the term *unnamed buffer* in 7
5.35.7.1. 7

The following commands shall be recognized in command mode:

`<newline>` Execute the current command line being edited.

`control-L` Redraw the current command line. Position the cursor at the same location on the new command line.

`#` Insert the character # at the beginning of the current command line and treat the current command line as a comment. This 8
line shall be entered into the command history; see 5.12. 8

`=` Display the possible shell word expansions (see 3.6) of the bigword at the current command-line position. These expansions shall be displayed on subsequent terminal lines. If the bigword contains none of the characters `?`, `*`, or `[`, an asterisk (`*`) shall be implicitly assumed at the end. If any directories are matched, these expansions shall have a `/` character appended. After the expansion, the line shall be redrawn, the cursor shall be repositioned at the current cursor position, and `sh` shall be placed in command mode.

`\` Perform pathname expansion (see 3.6.6) on the current bigword, up to the largest set of characters that can be matched uniquely. If the bigword contains none of the characters `?`, `*`, or `[`, an asterisk (`*`) shall be implicitly assumed at the end. This maximal expansion then shall replace the original bigword in the command line, and the cursor shall be placed after this expansion. If the resulting bigword completely and uniquely matches a directory, a `/` character shall be inserted directly after the bigword. If some other file is completely matched, a single `<space>` character shall be inserted after the bigword. After this operation, `sh` shall be placed in insert mode.

`*` Perform pathname expansion on the current bigword and insert all expansions into the command to replace the current bigword, with each expansion separated by a single `<space>` character. If at the end of the line, the current cursor position shall be moved to the first column position following the expansions and `sh` shall be placed in insert mode. Otherwise, the current cursor position shall be the last column position of the first character after the expansions and `sh` shall be placed in insert mode. If the current bigword contains none of the characters `?`, `*`, or `[`, before the operation, an asterisk shall be implicitly assumed at the end.

@letter Insert the value of the alias named *_letter*. The symbol *letter* represents a single alphabetic character from the portable character set; implementations may support additional characters as an extension. If the alias *letter* contains other editing commands, these commands shall be performed as part of the insertion. If no alias *letter* is enabled, this command shall have no effect.

[*count*]~ Convert, if the current character is a lowercase letter, to the equivalent uppercase letter, and vice versa, as prescribed by the current locale. The current cursor position then shall be advanced by one character. If the cursor was positioned on the last character of the line, the case conversion shall occur, but the cursor shall not advance. If the ~ command is preceded by a *count*, that number of characters shall be converted, and the cursor shall be advanced to the character position after the last character converted. If the *count* is larger than the number of characters after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count*]`.` Repeat the most recent nonmotion command, even if it was executed on an earlier command line. If the previous command was preceded by a *count*, and no count is given on the `.` command, the count from the previous command shall be included as part of the repeated command. If the `.` command is preceded by a *count*, this shall override any *count* argument to the previous command. The *count* specified in the `.` command shall become the count for subsequent `.` commands issued without a count.

[*number*]`v` Invoke the `vi` editor to edit the current command line in a temporary file. When the editor exits, the commands in the temporary file shall be executed. If a *number* is prefixed to the command, it specifies the command number in the command history to be edited, rather than the current command line.

[*count*]`l` (ell)
[*count*]`<space>`
Move the current cursor position to the next character position. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the *count* is larger than the number of characters after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count*]`h` Move the current cursor position to the *count*-th (default 1) previous character position. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the count is larger than the number of characters before the cursor, this shall not be considered an

error; the cursor shall move to the first character on the line.

[*count*]`w` Move to the start of the next word. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the *count* is larger than the number of words after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count*]`W` Move to the start of the next bigword. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the *count* is larger than the number of bigwords after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count*]`e` Move to the end of the current word. If at the end of a word, move to the end of the next word. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the *count* is larger than the number of words after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count*]`E` Move to the end of the current bigword. If at the end of a bigword, move to the end of the next bigword. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the *count* is larger than the number of bigwords after the cursor, this shall not be considered an error; the cursor shall advance to the last character on the line.

[*count*]`b` Move to the beginning of the current word. If at the beginning of a word, move to the beginning of the previous word. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the *count* is larger than the number of words preceding the cursor, this shall not be considered an error; the cursor shall return to the first character on the line.

[*count*]`B` Move to the beginning of the current bigword. If at the beginning of a bigword, move to the beginning of the previous bigword. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the *count* is larger than the number of bigwords preceding the cursor, this shall not be considered an error; the cursor shall return to the first character on the line.

`^` Move the current cursor position to the first character on the input line that is not a `<blank>`.

`$` Move to the last character position on the current command line.

`0` (Zero.) Move to the first character position on the current command line.

`[`*count*`]|` Move to the *count*-th character position on the current command line. If no number is specified, move to the first position. The first character position shall be numbered 1. If the count is larger than the number of characters on the line, this shall not be considered an error; the cursor shall be placed on the last character on the line.

`[`*count*`]f`*c* Move to the first occurrence of the character *c* that occurs after the current cursor position. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the character *c* does not occur in the line after the current cursor position, the terminal shall be alerted and the cursor shall not be moved.

`[`*count*`]F`*c* Move to the first occurrence of the character *c* that occurs before the current cursor position. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the character *c* does not occur in the line before the current cursor position, the terminal shall be alerted and the cursor shall not be moved.

`[`*count*`]t`*c* Move to the character before the first occurrence of the character *c* that occurs after the current cursor position. If the cursor was positioned on the last character of the line, the terminal shall be alerted and the cursor shall not be advanced. If the character *c* does not occur in the line after the current cursor position, the terminal shall be alerted and the cursor shall not be moved.

`[`*count*`]T`*c* Move to the character after the first occurrence of the character *c* that occurs before the current cursor position. If the cursor was positioned on the first character of the line, the terminal shall be alerted and the cursor shall not be moved. If the character *c* does not occur in the line before the current cursor position, the terminal shall be alerted and the cursor shall not be moved.

`[`*count*`];` Repeat the most recent `f`, `F`, `t`, or `T` command. Any number argument on that previous command shall be ignored. Errors are those described for the repeated command.

`[`*count*`],` Repeat the most recent `f`, `F`, `t`, or `T` command, ignoring the number argument to that command. However, reverse the direction of that command.

`a` Enter insert mode after the current cursor position. Characters that are entered shall be inserted before the next character.

`A` Enter insert mode after the end of the current command line.

`i` Enter insert mode at the current cursor position. Characters that are entered shall be inserted before the current character.

`I` Enter insert mode at the beginning of the current command line.

`R` Enter insert mode, replacing characters from the command line beginning at the current cursor position.

[*count*]`c` *motion*

Delete the characters between the current cursor position and the cursor position that would result from the specified *motion* command. Then enter insert mode before the first character following any deleted characters. If *count* is specified, it shall be applied to the *motion* command. A *count* shall be ignored for the motion commands:

`0` `^` `$` `c`

If the *motion* command is the character `c`, the current command line shall be cleared and insert mode shall be entered. If the *motion* command would move the current cursor position toward the beginning of the command line, the character under the current cursor position shall not be deleted. If the motion command would move the current cursor position toward the end of the command line, the character under the current cursor position shall be deleted. If the *count* is larger than the number of characters between the current cursor position and the end of the command line toward which the motion command would move the cursor, this shall not be considered an error; all of the remaining characters in the aforementioned range shall be deleted and insert mode shall be entered. If the motion command is invalid, the terminal shall be alerted, the cursor shall not be moved, and no text shall be deleted.

`C` Delete from the current character to the end of the line and enter insert mode at the new end of line.

`S` Clear the entire current command line and enter insert mode.

[*count*]`r`*c* Replace the current character with the character *c*. With a number *count*, replace the current and the following *count* −1 characters. After this command, the current cursor position shall be on the last character that was changed. If the *count* is larger than the number of characters after the cursor, this shall not be considered an error; all of the remaining characters shall be changed.

[*count*]`_` Append a `<space>` after the current character position and then append the last bigword in the previous input line after the `<space>`. Then enter insert mode after the last character just appended. With a number *count*, append the *count*-th bigword in the previous line.

[*count*]`x` Delete the character at the current cursor position and place the deleted character(s) in the save buffer. If the cursor was positioned on the last character of the line, the character shall be deleted and the cursor position shall be moved to the previous character (the new last character). If the *count* is larger than the number of characters after the cursor, this shall not be considered an error; all the characters from the cursor to the end of the line shall be deleted.

[*count*]`X` Delete the character before the current cursor position and place the deleted character(s) in the save buffer. The character under the current cursor position shall not change. If the cursor was positioned on the first character of the line, the terminal shall be alerted, and the `X` command shall have no effect. If the line contained a single character, the `X` command shall have no effect. If the line contained no characters, the terminal shall be alerted and the cursor shall not be moved. If the *count* is larger than the number of characters before the cursor, this shall not be considered an error; all the characters from before the cursor to the beginning of the line shall be deleted.

[*count*]`d` *motion*

Delete the characters between the current cursor position and the character position that would result from the *motion* command. A number *count* repeats the *motion* command *count* times. If the *motion* command would move toward the beginning of the command line, the character under the current cursor position shall not be deleted. If the *motion* command is `d`, the entire current command line shall be cleared. If the *count* is larger than the number of characters between the current cursor position and the end of the command line toward which the motion command would move the cursor, this shall not be considered an error; all of the remaining characters in the aforementioned range shall be deleted. The deleted characters shall be placed in the save buffer.

`D` Delete all characters from the current cursor position to the end of the line. The deleted characters shall be placed in the save buffer.

[*count*]`y` *motion*

Yank (i.e., copy) the characters from the current cursor position to the position resulting from the *motion* command into a save buffer. A number *count* shall be applied to the *motion* command. If the *motion* command would move toward the beginning of the command line, the character under the current cursor position shall not be included in the set of yanked characters. If the *motion* command is `y`, the entire current command line shall be yanked into the save buffer. The current cursor position shall be unchanged. If the *count* is larger than the number of characters between the current cursor position and the end of the command line toward which the motion command would move the cursor, this shall not be considered an error; all of the remaining characters in the aforementioned range shall be yanked.

`Y` Yank the characters from the current cursor position to the end of the line into the save buffer. The current character position shall be unchanged.

[*count*]`p` Put a copy of the current contents of the save buffer after the current cursor position. The current cursor position shall be advanced to the last character put from the save buffer. A *count* shall indicate how many copies of the save buffer shall be put.

[*count*]`P` Put a copy of the current contents of the save buffer before the current cursor position. The current cursor position shall be moved to the last character put from the save buffer. A *count* shall indicate how many copies of the save buffer shall be put.

`u` Undo the last command that modified the text of the current command line.

`U` Undo all changes made to the current command line since first entering command mode on the line.

[*count*]`k`
[*count*]`–` Replace the current command line with the previous command line in the shell command history. The cursor shall be positioned on the first character of the new command. A count preceding the command shall have the same effect as executing the command *count* times. If a `k` or `-` command retreats past the maximum number of commands in effect for this shell (affected by the **HISTSIZE** environment variable), the terminal shall be alerted and the command shall have no effect.

[*count*]`j`
[*count*]`+` Replace the current command line with the next command line in the shell command history. The cursor shall be positioned on the first character of the new command. The command history 8
position shall be remembered, and any `k` or `–` command, or `j` or `+` command, shall decrement or increment that position and then shall fetch the line at the new position. If a `j` or `+` command advances past the most recent line in the history, the current command line shall be restored to the contents before the first `k`
or `–`. 7

[*number*]`G` Replace the current command line with the contents of the oldest command line stored in the shell command history. With a number *number* , replace the current command line with the contents of command *number* in the history.

`/`*string* `<newline>`
Move backward through the command history, searching for the

specified *string* , beginning with the previous command line. If it is not found, the current command line shall be unchanged. If it is found in a previous line, this command shall behave equivalently to a set of `k` commands to reach that line. If *string* begins with ^, the characters after the ^ shall be matched only at the beginning of a line.

`?`*string* `<newline>`
: Move forward through the command history, searching for the specified string. If it is not found, the current command line shall be unchanged. If the string is found in the current command line, the current cursor position shall be moved to the beginning of that string. If it is found in the history, this command shall behave equivalently to a set of `j` commands to reach that line. If *string* begins with ^, the characters after the ^ shall be matched only at the beginning of a line.

`n`
: Repeat the most recent / or ? command.

`N`
: Repeat the most recent / or ? command, reversing the direction of the search.

4.56.10 sh Rationale. *(This subclause is not a part of P1003.2a)*

The default messages for the various **MAIL**-related messages are unspecified because they vary across implementations. Typical messages are:

`you have mail\n` or `you have new mail\n`

It is important that the descriptions of command line editing refer to the same shell as that in the base standard so that interactive users can also be application programmers without having to deal with programmatic differences in their two environments. It is also essential that the utility name `sh` be specified because this explicit utility name is too firmly rooted in historical practice of application programs for it to change.

Consideration was given to mandating a diagnostic message when attempting to set vi-mode on terminals that do not support command line editing. However, it is not historical practice for the shell to be cognizant of all terminal types and thus be able to detect inappropriate terminals in all cases. Implementations are encouraged to supply diagnostics in this case whenever possible, rather than leaving the user in a state where editing commands work incorrectly.

In earlier drafts, the KornShell-derived `emacs` mode of command-line editing was included, even though the `emacs` editor itself was not. The community of `emacs` editing enthusiasts was quite adamant that the full `emacs` editor not be included in POSIX. 2, because it was concerned that an attempt to standardize this very powerful environment would encourage vendors to ship versions conforming strictly to the standard, but lacking the extensibility required by the community.

The author of the original `emacs` program also expressed his desire to omit the program. Furthermore, there were a number of historical systems that did not include `emacs`, or included it without supporting it, but there were very few that did not include and support `vi`. The shell `emacs` command-line editing mode was finally omitted from this standard because it became apparent that the KornShell version and the editor being distributed with the GNU system had diverged in some respects. The author of `emacs` requested that the POSIX `emacs` mode either be deleted or have a significant number of unspecified conditions. Although the KornShell author agreed to consider changes to bring the shell into alignment, the developers of the standard decided to defer specification at this time, rather than attempting to agree on a specific subset of `emacs` late within balloting of the standard. (Furthermore, the standard developers included few individuals who had direct experience with `emacs` mode and the specification did not achieve the level of review afforded to the `vi` mode.) It is assumed that the `emacs` and KornShell developers will converge on a definition acceptable to both groups and this may be used as a model for a future version of this standard. In the interim, implementations are free to offer additional command-line editing modes based on the exact models of editors their users are most comfortable with.

4.70 `wait` — Await Process Completion

⇒ **4.70 `wait` Operands.** *In subclause 4.70, modify the description of the pid operand in the Operands subclause:*

pid One of the following:

(1) *<current contents>*

(2) On systems supporting both the POSIX.1 {8} job control option and the User Portability Utilities Option, a job control job ID (see 2.2.2.206) that identifies a background process group to be waited for. The job control job ID notation is applicable only for invocations of `wait` in the current shell execution environment; see 3.12. The exit status of `wait` shall be determined by the last command in the pipeline.

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

This additional syntax for *pid* is applicable only to systems having the job control features of the User Portability Extension. Thus, it is described in this supplement document, rather than in the base POSIX.2 document. See the rationale for `kill` for additional information on the effects of the job control job ID notation outside the current shell execution environment.

⇒ **4.70.10 `wait` Rationale.** *In subclause 4.70, add the following example to the rationale.*

Example: If the following sequence of commands is run in less than 31 seconds:

```
sleep 257 | sleep 31 &
jobs -l %%
```

either of the following commands will return the exit status of the second `sleep` in the pipeline:

```
wait <pid of sleep 31>
wait %%
```

Section 5: User Portability Utilities Option

This section describes the utilities that shall be implemented on all systems that claim conformance to the User Portability Utilities Option. Certain utilities are noted as having features that cannot be implemented on all terminal types; if the {POSIX2_CHAR_TERM} option is supported, the system shall support all such features on at least one terminal type. See 2.14.

Some of the utilities are only required on systems that also support the Software Development Utilities Option, the character-at-a-time terminal option (see 2.14) or that support the POSIX.1 {8} job control option; such utilities are so marked, directly before their Synopsis subclauses. All of the other utilities in this section are required only on systems that claim conformance to the User Portability Utilities Option.

User Portability Utilities Option Rationale. *(This subclause is not a part of P1003.2a)* .

All of the new utilities defined in the UPE are collected into a single large section. This was done for the practical reason that empty sections (chapters) were being held open for this work in the base POSIX.2 document. It was considered impractical to have enough empty chapters to handle all of the permutations of UPE, software development, and special terminal characteristics.

A summary of the combinations of options required for certain utilities in this section (where • indicates that the entire utility is optional unless the feature is provided and text indicates the portions that are optional):

Utility	{_POSIX2_UPE}	{_POSIX2_SW_DEV}	{_POSIX_JOB_CONTROL}	{_POSIX2_CHAR_TERM}
`bg`	•		•	
`ex`	•	`-t` option		see NOTE
`ctags`	•	•		
`fg`	•		•	
`more`	•	`-t` option		see NOTE
`nm`	•	•		
`sh`	•			see NOTE
`talk`	•			see NOTE
`vi`	•	`-t` option		•

NOTE: The `ex`, `more`, `sh`, and `talk` utilities also have terminal-related requirements that are allowed to result in error messages on some terminals (or all terminals, if {POSIX2_CHAR_TERM} is not supported); see their descriptions.

5.1 `alias` — Define or display aliases

5.1.1 Synopsis

`alias` [*alias-name* [=*string*] . . .]

5.1.2 Description

The `alias` utility shall create or redefine alias definitions or write the values of existing alias definitions to standard output. An alias definition provides a string value that shall replace a command name when it is encountered. See 3.3.1.

An alias definition shall affect the current shell execution environment and the execution environments of the subshells of the current shell. When used as specified by this standard, the alias definition shall not affect the parent process of the current shell nor any utility environment invoked by the shell. See 3.12.

5.1.3 Options

None.

5.1.4 Operands

The following operands shall be supported by the implementation:

alias-name Write the alias definition to standard output.

alias-name =*string*
Assign the value of *string* to the alias *alias-name* .

If no operands are given, all alias definitions shall be written to standard output.

5.1.5 External Influences

5.1.5.1 Standard Input

None.

5.1.5.2 Input Files

None.

5.1.5.3 Environment Variables

The following environment variables shall affect the execution of `alias`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.1.5.4 Asynchronous Events

Default.

5.1.6 External Effects

5.1.6.1 Standard Output

The format for displaying aliases (when no operands or only *name* operands are specified) shall be:

`"%s=%s\n"`, *name* , *value*

The *value* string shall be written with appropriate quoting so that it is suitable 7
for re-input to the shell. See the description of shell quoting in 3.2. 7

5.1.6.2 Standard Error

Used only for diagnostic messages.

5.1.6.3 Output Files

None.

5.1.7 Extended Description

None.

5.1.8 Exit Status

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

0 Successful completion.

>0 One of the *name* operands specified did not have an alias definition, or an error occurred.

5.1.9 Consequences of Errors

Default.

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

Usage, Examples

None.

History of Decisions Made

The `alias` description is based on that from historical KornShell implementations. Known differences exist between that and the C-Shell. The KornShell version was adopted to be consistent with all the other KornShell features in POSIX. 2, such as command-line editing.

Since `alias` affects the current shell execution environment, it is generally provided as a shell regular built-in.

Historical versions of the KornShell have allowed aliases to be exported to scripts that are invoked by the same shell. This is triggered by the `alias -x` flag; it is allowed by POSIX. 2 only when an explicit extension such as `-x` is used. The working group felt that aliases were of use primarily to interactive users and that they should normally not affect shell scripts called by those users; functions are available to such scripts.

Historical versions of the KornShell have not written aliases in a quoted manner
suitable for reentry to the shell, but the base POSIX. 2 standard has made this a 7
requirement for all similar output. Therefore, consistency with the base standard 7
was chosen over this minor detail of historical practice. 7

5.2 `at` — Execute commands at a later time

5.2.1 Synopsis

`at` [`−m`] [`−f` *file*] [`−q` *queuename*] `−t` *time*

`at` [`−m`] [`−f` *file*] [`−q` *queuename*] *timespec* ...

`at` `−r` *at_job_id* ...

`at` `−l` `−q` *queuename*

`at` `−l` [*at_job_id* ...]

5.2.2 Description

The `at` utility shall read commands from standard input and group them together as an *at-job*, to be executed at a later time.

The at-job shall be executed in a separate invocation of the shell, running in a separate process group with no controlling terminal, except that the environment variables, current working directory, file creation mask, and other implementation-defined execution-time attributes in effect when the `at` utility is executed shall be retained and used when the at-job is executed.

When the at-job is submitted, the *at_job_id* and scheduled time shall be written to standard error. The *at_job_id* is an identifier that shall be a string consisting solely of alphanumeric characters and the period character. The *at_job_id* shall be assigned by the system when the job is scheduled such that it uniquely identifies a particular job.

User notification and the processing of the job's standard output and standard error are described under the `−m` option.

5.2.3 Options

The `at` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`−f` *file* Specify the pathname of a file to be used as the source of the at-job, instead of standard input.

`−l` (The letter ell.) Report all jobs scheduled for the invoking user if no *at_job_id* operands are specified. If *at_job_id* s are specified, report only information for these jobs. The output shall be written to standard output.

`−m` Send mail to the invoking user after the at-job has run, announcing its completion. Standard output and standard error produced by the at-job shall be mailed to the user as well, unless redirected elsewhere. Mail shall be sent even if the job produces no output.

If `−m` is not used, the job's standard output and standard error 8
shall be provided to the user via an implementation-defined 8
mechanism, unless they are redirected elsewhere; if there is no 8
such output to provide, the implementation need not notify the 8
user of the job's completion. 8

`−q` *queuename*

Specify in which queue to schedule a job for submission. When used with the `−l` option, limit the search to that particular queue. By default, at-jobs shall be scheduled in queue ‘`a`’’. In contrast, queue ‘`b`’’ shall be reserved for batch jobs. (See the `batch` utility
in 5.3.) The meanings of all other *queuenames* are implementa- 7
tion defined. 7

`−r` Remove the jobs with the specified *at_job_id* operands that were previously scheduled by the `at` utility.

`−t` *time* Submit the job to be run at the time specified by the *time* option-argument, which shall have the format as specified by the `touch` (see 4.63) utility.

5.2.4 Operands

The following operands shall be supported by the implementation:

at_job_id

The name reported by a previous invocation of the `at` utility at the time the job was scheduled.

8

timespec

Submit the job to be run at the date and time specified. All of the 8
timespec operands shall be concatenated into the equivalent of a single 8
string with `<space>` separation added between the operands and this 8
string shall be parsed as described in the grammar at the end of this 8
subclause. The date and time shall be interpreted as being in the time 8
zone of the user (as determined by the **TZ** variable), unless a time-zone 8
name appears as part of *time* , below. 8

In the POSIX Locale, the following describes the three parts of the time 8
specification string. All of the values from the LC_TIME categories in the POSIX Locale shall be recognized in a case-insensitive manner.

time The *time* can be specified as one, two, or four digits. One- and two-digit numbers shall be taken to be hours, four-digit numbers to be hours and minutes. The time can alternately be specified as two numbers separated by a colon, meaning *hour* `:` *minute* . An AM/PM indication (one of the values from the `am_pm` keywords in the LC_TIME locale category) can follow the time; otherwise, a 24-hour clock time shall be understood. A time-zone name can also follow to further qualify the time. The acceptable time-zone names are implementation defined, except that they shall be case-insensitive and the string `"utc"` shall be supported to indicate the time is in Coordinated Universal Time. The *time* field can also be one of the following tokens in the POSIX Locale:

- `midnight`
  Indicates the time 12:00 am (00:00).
- `noon` Indicates the time 12:00 pm.
- `now` Indicate the current day and time. Invoking `at` <*now*> shall submit an at-job for potentially immediate execution (i.e., subject only to unspecified scheduling delays).

date An optional *date* can be specified as either a month name (one of the values from the `mon` or `abmon` keywords in the LC_TIME locale category) followed by a day number (and possibly year number preceded by a comma) or a day of the week (one of the values from the `day` or `abday` keywords in the LC_TIME locale category). Two special ‘‘days’’ shall be recognized in the POSIX Locale:

- `today`
  Indicates the current day.
- `tomorrow`
  Indicates the day following the current day.

If no *date* is given, `today` shall be assumed if the given time is greater than the current time and `tomorrow` shall be assumed if it is less. If the given month is less than the current month (and no year is given), next year shall be assumed.

increment
The optional *increment* shall be a number preceded by a plus sign (+) and suffixed by one of the following: `minutes`, `hours`, `days`, `weeks`, `months`, or `years`. (The singular forms shall be also accepted.) The keyword `next` shall be equivalent to an increment number of `+ 1`. For example, the following are equivalent commands:

```
                     at 2pm + 1 week
                     at 2pm next week
```
8
8

The following grammar describes the precise format of *timespec* in the POSIX 7
Locale. The general conventions for this style of grammar are described in 2.1.2. 7
Any discrepancies found between this grammar and the preceding description 7
shall be resolved in favor of the grammar. The longest possible token or delimiter 8
shall be recognized at a given point. White space shall also delimit tokens. 8

```
%token hr24clock_hr_min
%token hr24clock_hour
/*
  A hr24clock_hr_min is a one, two, or four digit number.
  A one or two digit number constitutes a hr24clock_hour.
  A hr24clock_hour may be any of the single digits '0' - '9',
  or may be double digits, ranging from "00" - "23".
  If a hr24clock_hr_min is a four digit number, the first
  two digits must be a valid hr24clock_hour, while the last
  two represent the number of minutes, from "00" - "59".
*/
%token wallclock_hr_min
%token wallclock_hour
/*
  A wallclock_hr_min is a one, two or four digit number.
  A one or two digit number constitutes a wallclock_hour.
  A wallclock_hour may be any of the single digits '1' - '9',
  or may be double digits, ranging from "01" - "12".
  If a wallclock_hr_min is a four digit number, the first two
  digits must be a valid wallclock_hour, while the last two
  represent the number of minutes, from "00" - "59".
*/
%token minute
/*
  A minute is a one or two digit number whose values can be
  '0' - '9' or "00" - "59".
*/
%token day_number
/*
  A day_number is a number in the range appropriate for the
  particular month and year specified by month_name and
  year_number, respectively.  If no year_number is given,
  the current year is assumed if the given date and time
  are later this year.  If no year_number is given and the
  date and time have already occurred this year and the
  month is not the current month, next year is the assumed year.
*/
%token year_number
/*
  A year_number is a four-digit number representing the
  year A.D., in which the at job is to be run.
```

```
*/
%token inc_number
/*
  The inc_number is the number of times the succeeding increment
  period is to be added to the specified date and time.
*/
%token timezone_name
/*
  The name of an optional time zone suffix to the time                8
  field, in an implementation-defined format.                         8
*/
%token month_name
/*
  One of the values from the "mon" or "abmon" keywords in
  the LC_TIME locale category.
*/
%token day_of_week
/*
  One of the values from the "day" or "abday" keywords in
  the LC_TIME locale category.
*/
%token am_pm
/*
  One of the values from the "am_pm" keyword in
  the LC_TIME locale category.
*/
%start timespec
%%
timespec      : time
              | time date
              | time increment
              | time date increment
              | nowspec                                               8
              ;
nowspec       : "now"
              | "now" increment
              ;
time          : hr24clock_hr_min
              | hr24clock_hr_min timezone_name
              | hr24clock_hour ":" minute
              | hr24clock_hour ":" minute timezone_name
              | wallclock_hr_min am_pm
              | wallclock_hr_min am_pm timezone_name
              | wallclock_hour ":" minute am_pm
              | wallclock_hour ":" minute am_pm timezone_name
              | "noon"
              | "midnight"
                                                                      8
```

```
                ;
date            : month_name day_number
                | month_name day_number "," year_number
                | day_of_week
                | "today"
                | "tomorrow"
                ;
increment       : "+" inc_number inc_period
                : "next" inc_period                              8
                ;
inc_period      : "minute" | "minutes"
                | "hour" | "hours"
                | "day" | "days"
                | "week" | "weeks"
                | "month" | "months"
                | "year" | "years"
                ;
```

5.2.5 External Influences

5.2.5.1 Standard Input

The standard input shall be a text file consisting of commands acceptable to the shell command language described in Section 3. The standard input shall only be used if no `-f` *file* option is specified. 8

5.2.5.2 Input Files

See Standard Input.

5.2.5.3 Environment Variables

The following environment variables shall affect the execution of `at`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).

LC_MESSAGES This variable shall determine the language in which messages should be written.

LC_TIME This variable shall determine the the format and contents for date and time strings written and accepted by `at`.

SHELL This variable shall be interpreted as a name of a com- 7
mand interpreter to be used to invoke the at-job. If the 7
variable is unset or null, `sh` (see 4.56) shall be used. If it 7
is set to a value other than a name for `sh`, the implemen- 7
tation shall do one of the following: use that shell; use `sh`; 7
use the login shell from the user database; any of the 7
preceding accompanied by a warning diagnostic about 7
which was chosen. 7

TZ This variable shall specify a time zone. The job shall be submitted for execution at the time specified by *timespec* or `−t` *time* relative to the time zone specified by the **TZ** variable. If *timespec* specifies a time zone, it shall override **TZ**. If *timespec* does not specify a time zone and **TZ** is unset or null, an unspecified default time zone shall be used.

5.2.5.4 Asynchronous Events

Default.

5.2.6 External Effects

5.2.6.1 Standard Output

When standard input is a terminal, prompts of unspecified format for each line of 8
the user input described in 5.2.5.1 may be written to standard output. 8

In the POSIX Locale, the following shall be written to the standard output for each job when jobs are listed in response to the `−l` option:

`"%s\t%s\n"`, *at_job_id* , *<date>*

where *<date>* shall be equivalent in format to the output of:

`date +"%a %b %e %T %Y"` 7

The date and time written shall be adjusted so that they appear in the time zone of the user (as determined by the **TZ** variable).

5.2.6.2 Standard Error

The following shall be written to standard error when a job has been successfully submitted:

`"job %s at %s\n"`, *at_job_id* , *<date>*

where *<date>* shall have the same format as is described in Standard Output. Neither this, nor warning messages concerning the selection of the command interpreter, shall be considered a diagnostic that changes the exit status.

Diagnostic messages, if any, shall be written to standard error.

5.2.6.3 Output Files

None.

5.2.7 Extended Description

None.

5.2.8 Exit Status

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

- 0 The `at` utility successfully submitted, removed, or listed a job or jobs.
- >0 An error occurred.

5.2.9 Consequences of Errors

The job shall not be scheduled, removed, or listed.

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

Usage, Examples

The `at` utility reads from standard input the commands to be executed at a later
time. It may be useful to redirect standard output and standard error within the 7
specified commands. 7

The format of the `at` command line shown here is guaranteed only for the POSIX 8
Locale. Other cultures may be supported with substantially different interfaces, 8
although implementations are encouraged to provide comparable levels of func- 8
tionality. 8

This sequence can be used at a terminal:

```
at -m 0730 tomorrow
sort < file >outfile
EOT
```

This sequence, which demonstrates redirecting standard error to a pipe, is useful in a command procedure (the sequence of output redirection specifications is significant):

```
at now + 1 hour <<!
diff file1 file2 2>&1 >outfile | mailx mygroup
!
```

To have a job reschedule itself, `at` can be invoked from within the at-job.

The spacing of the three portions of the POSIX Locale *timespec* is quite flexible as 8
long as there are no ambiguities. Examples of various times and operand presen- 8
tation include: 8

```
at 0815am Jan 24                 8
at 8 :15amjan24                  8
at now "+ 1day"                  8
at 5 pm FRIday                   8
at '17                           8
        utc+                     8
        30minutes'               8
```

Since the commands run in a separate shell invocation, running in a separate process group with no controlling terminal, open file descriptors, traps, and priority inherited from the invoking environment are lost.

Some implementations do not allow substitution of different shells using **SHELL** . 7
System V systems, for example, have used the login shell value for the user in 7
`/etc/passwd`. To reliably select another command interpreter, the user must 7
include it as part of the script, such as: 7

```
$ at 1800                        7
myshell myscript                 7
job ... at ...                   7
$                                7
```

History of Decisions Made

Due to various implementations, and since the POSIX system administration group is dealing with administrative issues, the location or use of the files `at.allow` and `at.deny` were omitted. The following, if using these files, is a suggested implementation:

> Users shall be permitted to use `at` if their name appears in the file `at.allow`. If that file does not exist, the file `at.deny` shall be checked to determine if the user shall be denied access to `at`. If neither file exists, only root shall be allowed to submit a job. If only `at.deny` exists and is empty, global usage shall be permitted. The allow/deny files shall consist of one user name per line.

The `−t` *time* option was added as a new capability to support an internationalized way of specifying a time for execution of the submitted job.

Early drafts added a ‘‘jobname’’ concept as a way of giving submitted jobs names that are meaningful to the user submitting them. The historical, system-specified *at_job_id* gives no indication of what the job is. Upon further reflection, it was decided that the benefit of this was not worth the change in historical interface. It is anticipated that considerably more sophisticated ways of controlling background or batch work will be the subject of future revisions to this standard.

The −q option historically has been an undocumented option, used mainly by the batch utility.

The System V −m option was added to provide a method for informing users that an at-job had completed. Otherwise, users are only informed when output to standard error or standard output are not redirected. This information is mailed to the user when the −m option is provided. When −m is not given, most systems mail the standard output and standard error to the user, but this is not required; some other method, such as a log file, could be used.

The behavior of at <*now*> was changed in an earlier draft from being unspecified to submitting a job for potentially immediate execution. Historical BSD at implementations support this. Historical System V implementations give an error in that case, but a change to the System V versions should have no backward compatibility ramifications.

On BSD-based systems, a −u *user* option has allowed those with appropriate privileges to access the work of other users. Since this is primarily a system administration feature and is not universally implemented, it has been omitted. Similarly, a specification for the output format for user with appropriate privileges viewing the queues of other users has been omitted.

The −f *file* option from System V is used instead of the BSD method of using the last operand as the pathname. The BSD method is ambiguous—does

```
at 1200 friday
```

mean the same thing if there is a file named friday in the current directory?

The *at_job_id* is composed of a limited character set in historical practice and it is mandated here to invalidate systems that might try using characters that require shell quoting or could not be easily parsed by shell scripts.

The at utility varies between System V and BSD systems in the way time zones are used. On System V systems, the **TZ** variable affects the at-job submission times and the times displayed for the user. On BSD systems, **TZ** is not taken into account. The BSD behavior is easily achieved with the current specification. If the user wishes to have the time zone default to that of the system, she merely needs to issue the at command immediately following an unsetting or null assignment to **TZ**, For example,

```
TZ= at noon . . .
```

will give the desired BSD result.

A previous draft indicated that various signals would be ignored. What it meant to say was that the at-job would not be affected by the various keyboard-related signals the user could initiate. The current wording about no controlling terminal should now suffice.

While the yacc-like grammar specified in 5.2.4 is lexically unambiguous with respect to the digit strings, a lexical analyzer would probably be written to look for and return digit strings in those cases. The parser could then check whether the digit string returned is a valid `day_number`, `year_number`, etc., based on the context.

5.3 `batch` — Execute commands when the system load permits

5.3.1 Synopsis

```
batch
```

5.3.2 Description

The `batch` utility shall read commands to be executed at a later time. It shall be the equivalent of the command:

```
at -q b -m now
```
7

where queue `b` is a special `at` queue, specifically for batch jobs. Batch jobs shall
be submitted to the batch queue with no time constraints and shall be run by the
system using algorithms, based on unspecified factors, that may vary with each 7
invocation of `batch`.

5.3.3 Options

None.

5.3.4 Operands

None.

5.3.5 External Influences

5.3.5.1 Standard Input

See 5.2.

5.3.5.2 Input Files

None.

5.3.5.3 Environment Variables

See 5.2.

5.3.5.4 Asynchronous Events

See 5.2.

5.3.6 External Effects

5.3.6.1 Standard Output

See 5.2. 8

5.3.6.2 Standard Error

See 5.2.

5.3.6.3 Output Files

None.

5.3.7 Extended Description

None.

5.3.8 Exit Status

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

0 The `batch` utility successfully submitted a job.

>0 An error occurred.

5.3.9 Consequences of Errors

The job shall not be scheduled.

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

Usage, Examples

The `batch` utility reads from standard input the commands to be executed at a later time. It may be useful to redirect standard output within the specified commands.

This sequence can be used at a terminal:

```
batch
sort < file >outfile
EOT
```

This sequence, which demonstrates redirecting standard error to a pipe, is useful in a command procedure (the sequence of output redirection specifications is significant):

```
batch <<!
diff file1 file2 2>&1 >outfile | mailx mygroup
!
```

History of Decisions Made

Early drafts described `batch` in a manner totally separated from `at`, even though the historical model treated it almost as a synonym for `at -q b`. A number of features were added to list and control batch work separately from those in `at`. Upon further reflection, it was decided that the benefit of this was not worth the change in historical interface. It is anticipated that considerably more sophisticated ways of controlling background or batch work will be the subject of future revisions to this standard.

The −`m` option was included on the equivalent `at` command because it is historical 7
practice to mail results to the submitter, even if all job-produced output is 7
redirected. As explained in the rationale for `at`, the `now` keyword submits the job 7
for immediate execution (after scheduling delays), despite some historical systems 7
where `at now` would have been considered an error. 7

5.4 bg — Run jobs in the background

This utility shall be provided on systems that support both the User Portability 7
Utilities Option and the POSIX.1 {8} job control option. On other systems it is 7
optional. 7

5.4.1 Synopsis

bg [*job_id* ...]

5.4.2 Description

If job control is enabled (see the description of `set −m` in 3.14.11), the `bg` utility shall resume suspended jobs from the current environment (see 3.12) by running them as background jobs. If the job specified by *job_id* is already a running background job, the `bg` utility shall have no effect and shall exit successfully.

Using `bg` to place a job into the background shall cause its process ID to become ‘‘known in the current shell execution environment,’’ as if it had been started as an asynchronous list; see 3.9.3.1.

5.4.3 Options

None.

5.4.4 Operands

The following operand shall be supported by the implementation:

job_id Specify the job to be resumed as a background job. If no *job_id* operand is given, the most recently suspended job shall be used. The format of *job_id* is described in 2.2.2.206.

5.4.5 External Influences

5.4.5.1 Standard Input

None.

5.4.5.2 Input Files

None.

5.4.5.3 Environment Variables

The following environment variables shall affect the execution of `bg`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.4.5.4 Asynchronous Events

Default.

5.4.6 External Effects

5.4.6.1 Standard Output

The output of `bg` shall consist of a line in the format:

`"[%d] %s\n"`, <*job-number*> , <*command*>

where the fields are as follows:

<*job-number* > A number that can be used to identify the job to the `wait`, `fg`, and `kill` utilities. Using these utilities, the job can be identified by prefixing the job number with `%`.

<*command* > The associated command that was given to the shell.

5.4.6.2 Standard Error

Used only for diagnostic messages.

5.4.6.3 Output Files

None.

5.4.7 Extended Description

None.

5.4.8 Exit Status

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

0	Successful completion.
>0	An error occurred.

5.4.9 Consequences of Errors

If job control is disabled, the `bg` utility shall exit with an error and no job shall be placed in the background.

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

Usage, Examples

The `bg` utility will not work as expected when it is operating in its own utility execution environment because that environment will have no suspended jobs. In the following examples:

```
... | xargs bg
(bg)
```

each `bg` operates in a different environment and will not share its parent shell's understanding of jobs. For this reason, `bg` is generally implemented as a shell regular built-in.

History of Decisions Made

The extensions to the shell specified in POSIX.2 and POSIX.2a have mostly been based on features provided by the KornShell. The job control features provided by `bg`, `fg`, and `jobs` are also based on the KornShell. The developers of the standard examined the characteristics of the C-Shell versions of these utilities and found that differences exist. Despite widespread use of the C-Shell, the KornShell versions were selected for this standard to maintain a degree of uniformity with the rest of the KornShell features selected (such as the very popular command-line editing features).

8

The `bg` utility is expected to wrap its output if the output exceeds the number of display columns.

5.5 `crontab` — Schedule periodic background work

5.5.1 Synopsis

`crontab` [*file*]

`crontab [ −e | −l | −r ]`

5.5.2 Description

The `crontab` utility shall create, replace, or edit a user's *crontab* entry; a crontab entry is a list of commands and the times at which they are to be executed. The new crontab entry can be input by specifying *file* or input from standard input if no *file* operand is specified, or by using an editor, if `−e` is specified.

Upon execution of a command from a crontab entry, the implementation shall supply a default environment, defining at least the following environment variables:

HOME A pathname of the user's home directory.

LOGNAME The user's login name.

PATH A string representing a search path guaranteed to find all of the standard utilities.

SHELL A pathname of the command interpreter. When `crontab` is invoked as specified by this standard, the value shall be a pathname for `sh` (see 4.56).

The values of these variables when `crontab` is invoked as specified by this standard shall not affect the default values provided when the scheduled command is run.

If standard output and standard error are not redirected by commands executed from the crontab entry, any generated output or errors shall be mailed, via an implementation-defined method, to the user.

5.5.3 Options

The `crontab` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`−e` Edit a copy of the invoking user's crontab entry, or create an empty entry to edit if the crontab entry does not exist. When editing is complete, the entry shall be installed as the user's crontab entry.

`−l` (The letter ell.) List the invoking user's crontab entry.

`−r` Remove the invoking user's crontab entry.

5.5.4 Operands

The following operand shall be supported by the implementation:

file The pathname of a file that contains specifications, in the format defined in 5.5.5.2, for crontab entries.

5.5.5 External Influences

5.5.5.1 Standard Input

See Input Files.

5.5.5.2 Input Files

In the POSIX Locale, a crontab entry shall be a text file consisting of lines of six fields each. The fields shall be separated by `<blank>`s. The first five fields shall be integer patterns that specify the following:

(1) Minute (0−59)

(2) Hour (0−23)

(3) Day of the month (1−31)

(4) Month of the year (1−12)

(5) Day of the week (0−6 with 0=Sunday).

Each of these patterns can be either an asterisk (meaning all valid values), an element, or a list of elements separated by commas. An element shall be either a number or two numbers separated by a hyphen (meaning an inclusive range). The specification of days can be made by two fields (day of the month and day of the week). If month, day of month, and day of week are all asterisks, every day shall be matched. If either the month or day of month is specified as an element or list, but the day of week is an asterisk, the month and day of month fields shall specify the days that match. If both month and day of month are specified as asterisk, but day of week is an element or list, then only the specified days of the week shall match. Finally, if either the month or day of month is specified as an element or list, and the day of week is also specified as an element or list, then any day matching either the month and day of month, or the day of week, shall be matched.

The sixth field of a line in a crontab entry is a string that shall be executed by `sh` (see 4.56) at the specified times. A percent-sign character in this field shall be translated to a `<newline>` character. Any character preceded by a backslash (including the `%`) shall cause that character to be treated literally. Only the first line (up to a `%` or end of line) of the command field shall be executed by the command interpreter. The other lines shall be made available to the command as standard input.

Blank lines and those whose first non-`<blank>` character is # shall be ignored.

5.5.5.3 Environment Variables

The following environment variables shall affect the execution of `crontab`:

EDITOR This variable shall determine the editor to be invoked when the `−e` option is specified. The default editor shall be `vi`.

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.5.5.4 Asynchronous Events

Default.

5.5.6 External Effects

5.5.6.1 Standard Output

If the `−l` option is specified, the crontab entry shall be written to the standard output.

5.5.6.2 Standard Error

Used only for diagnostic messages.

5.5.6.3 Output Files

None.

5.5.7 Extended Description

None.

5.5.8 Exit Status

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

0 Successful completion.

>0 An error occurred.

5.5.9 Consequences of Errors

The user's crontab entry is not submitted, removed, edited, or listed.

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

Usage, Examples

As an example of specifying the two types of days,

```
0 0 1,15 * 1
```

would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to * (for example,

```
0 0 * * 1
```

would run a command only on Mondays).

The default settings of the **HOME**, **LOGNAME**, **PATH**, and **SHELL** variables that are given to the scheduled job are not affected by the settings of those variables when `crontab` is run; as stated, they are defaults. The text about ''invoked as specified by this standard'' means that the implementation may provide extensions that allow these variables to be affected at runtime, but that the user has to take explicit action in order to access the extension, such as give a new option flag or modify the format of the crontab entry.

History of Decisions Made

All references to a `cron` daemon and to `cron` *files* have been removed. Although historical implementations have used this arrangement, there is no reason to limit future implementations.

This description of `crontab` is designed to support only users with normal privileges. The format of the input is based on the System V `crontab`; however, there is no requirement here that the actual system database used by the `cron` daemon (or similar mechanism) use this format internally. For example, systems derived from BSD will likely have an additional field appended that indicates the user identity to be used when the job is submitted.

The `−e` option was adopted from the *SVID* as a user convenience, although it does not exist in all historical implementations.

5.6 `csplit` — Split files based on context

5.6.1 Synopsis

`csplit [−ks][−f` *prefix* `][−n` *number* `]` *file arg1* ... *argn*

5.6.2 Description

The `csplit` utility shall read the file named by the *file* operand, write all or part of that file into other files as directed by the *arg* operands, and write the sizes of the files.

5.6.3 Options

The `csplit` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`−f` *prefix* Name the created files *prefix* `00`, *prefix* `01`, ..., *prefixn* . The default shall be `xx00` ... `xx`*n*. If the *prefix* argument would create a filename exceeding {NAME_MAX} bytes, an error shall result, `csplit` shall exit with a diagnostic message and no files shall be created.

`−k` Leave previously created files intact. By default, `csplit` shall remove created files if an error occurs.

`−n` *number* Use *number* decimal digits to form filenames for the file pieces. The default shall be 2.

`−s` Suppress the output of file size messages.

5.6.4 Operands

The following operands shall be supported by the implementation:

file The pathname of a text file to be split. If *file* is −, the standard input shall be used.

The operands *arg1* . . . *argn* can be a combination of the following:

/*rexp*/[*offset*]
: Create a file using the content of the lines from the current line up to, but not including, the line that results from the evaluation of the regular expression with *offset*, if any, applied. The regular expression *rexp* shall follow the rules for basic regular expressions described in 2.8.3. The optional *offset* shall be a positive or negative integer value representing a number of lines. The integer value shall be preceded by + or −. If the selection of lines from an offset expression of this type would create a file with zero lines, or one with greater than the number of lines left in the input file, the results are unspecified. After the section is created, the current line shall be set to the line that results from the evaluation of the regular expression with any offset applied. The pattern match of *rexp* shall be always applied from the current line to the end of the file.

%*rexp*%[*offset*]
: This operand is the same as /*rexp*/[*offset*], except that no file shall be created for the selected section of the input file. 7

line_no Create a file from the current line up to (but not including) the line number *line_no*. Lines in the file shall be numbered starting at one. The current line becomes *line_no*.

{*num*} Repeat operand. This operand can follow any of the operands described previously. If it follows a *rexp* type operand, that operand shall be applied *num* more times. If it follows a *line_no* operand, the file shall be split every *line_no* lines, *num* times, from that point.

An error shall be reported if an operand does not reference a line between the current position and the end of the file.

5.6.5 External Influences

5.6.5.1 Standard Input

See Input Files.

5.6.5.2 Input Files

The input file shall be a text file.

5.6.5.3 Environment Variables

The following environment variables shall affect the execution of `csplit`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_COLLATE This variable shall determine the locale for character collation information in regular expressions.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files), and the behavior of character classes within regular expressions.

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.6.5.4 Asynchronous Events

If the `−k` option is specified, created files shall be retained. Otherwise the default action occurs.

5.6.6 External Effects

5.6.6.1 Standard Output

Unless the `−s` option is used, the standard output shall consist of one line per file created, with a format as follows:

`"%d\n"`, <*file size in bytes*>

5.6.6.2 Standard Error

Used only for diagnostic messages.

5.6.6.3 Output Files

The output files shall contain portions of the original input file, otherwise unchanged.

5.6.7 Extended Description

None.

5.6.8 Exit Status

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

- 0 Successful completion.
- >0 An error occurred.

5.6.9 Consequences of Errors

By default, created files shall be removed if an error occurs. When the `−k` option is specified, created files shall not be removed if an error occurs.

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

Usage, Examples

This example creates four files, `cobol00` ... `cobol03`:

```
csplit −f cobol file ’/procedure division/’ /par5./ /par16./
```

After editing the split files, they can be recombined as follows:

```
cat cobol0[0−3] > file
```

Note that this example overwrites the original file.

This example would split the file after the first 99 lines, and every 100 lines there 7
after, up to 9 999 lines; this is because lines in the file are numbered from 1 rather 7
than zero, for historical reasons. 7

```
csplit −k file 100 {99}
```

Assuming that `prog.c` follows the C-language coding convention of ending routines with a } at the beginning of the line, this example will create a file containing each separate C routine (up to 21) in `prog.c`.

```
csplit −k prog.c ’%main(%’ ’/^}/+1’ {20}
```

The use of repeated regular expressions with negative offsets produces bizarre results in historical implementations.

History of Decisions Made

The `−n` option was added to extend the range of filenames that could be handled.

Consideration was given to adding a `−a` flag to use the alphabetic file name generation used by the historical `split` utility, but the functionality added by the `−n` option was deemed to make alphabetic naming unnecessary.

5.7 `ctags` — Create a tags file

This utility shall be provided on systems that support the User Portability Utili- 7
ties Option, the Software Development Utilities Option, and either or both of the 7
C Language Development Utilities Option and FORTRAN Development Utilities 7
Option. On other systems it is optional. 7

5.7.1 Synopsis

`ctags [−ax][−f` *tagsfile* `]` *pathname* ...

5.7.2 Description

The `ctags` utility shall create a *tags* file from C-language or FORTRAN source file(s) specified by the *pathname* operands. The tags file shall list the locators of language-specific objects within the source files. A locator consists of a name, pathname, and either basic regular expression or a line number that can be used in searching for the object definition. The objects that shall be recognized are specified in 5.7.7.

5.7.3 Options

The `ctags` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`−a` Append to tags file.

`−f` *tagsfile* Write the object locator lists into *tagsfile* instead of the default file named `tags` in the current directory.

`−x` Produce a list of object names, the line number and file name in which each is defined, as well as the text of that line, and write this to the standard output.

5.7.4 Operands

The following *pathname* operands shall be supported by the implementation:

file `.c` Files with basenames ending with the `.c` suffix shall be treated as C-language source code. Such files that are not valid input to `c89` produce unspecified results.

file `.h` Files with basenames ending with the `.h` suffix shall be treated as C-language source code. Such files that are not valid input to `c89` produce unspecified results.

file `.f` Files with basenames ending with the `.f` suffix shall be treated as FORTRAN language source code. Such files that are not valid input to `fort77` produce unspecified results.

The handling of other files is implementation defined.

5.7.5 External Influences

5.7.5.1 Standard Input

See Input Files.

5.7.5.2 Input Files

The input files shall be text files containing source code in the language indicated by the operand filename suffixes.

5.7.5.3 Environment Variables

The following environment variables shall affect the execution of `ctags`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_COLLATE This variable shall determine the order in which output is sorted for the `-x` option. The POSIX Locale shall determine the order in which the tags file is written.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters within arguments and input files). When processing C-language source code, if the

locale is not compatible with the C locale described by the 7
C Standard {7}, the results are unspecified. 7

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.7.5.4 Asynchronous Events

Default.

5.7.6 External Effects

5.7.6.1 Standard Output

The list of object name information produced by the `-x` option shall be written to standard output in the following format:

`"%s %d %s %s"`, *<object-name>* , *<line-number>* , *<filename>* , *<text>* 8

where *<text>* is the text of line *<line-number>* of file *<filename>* .

5.7.6.2 Standard Error

Used only for diagnostic messages.

5.7.6.3 Output Files

The format of the output file shall be:

`"%s\t%s\t/%s/\n"`, *<identifier>* , *<filename>* , *<rexp>*

where *<rexp>* is a basic regular expression (see 2.8.3) that could be used by an 7
editor to find the defining instance of *<identifier>* in *<filename>* (where ‘‘defining 8
instance’’ is indicated by the declarations listed in 5.7.7). 8

An alternative format is: 8

`"%s\t%s\t%d\n"`, *<identifier>* , *<filename>* , *<lineno>* 8

where *<lineno>* is a decimal line number that could be used by an editor to find 7
<identifier> in *<filename>* . This alternative format shall not be produced by 7
`ctags` when it is used as described by this standard, but the standard utilities 7
that process tags files shall be able to process this format as well as the preceding 7
one. 7

In either format, the file shall be sorted by identifier, based on the collation 8
sequence in the POSIX Locale. 8

5.7.7 Extended Description

If the operand identifies C-language source, the `ctags` utility shall attempt to 7
produce an output line for each of the following objects: 7

— Function definitions 7

— Typedefs 7

— Macros with arguments 7

It may also produce output for any of the following objects: 7

— Function prototypes 7

— Structures 7

— Unions 7

— Global variable definitions 7

— Enums 7

— Macros without arguments 7

— `#define` statements 7

— `#line` statements 7

Any `#if` and `#ifdef` statements shall produce no output. The tag `main` is treated specially in C programs. The tag formed shall be created by prefixing `M` to the name of the file, with the trailing `.c`, and leading pathname components (if
any) removed. On systems that do not support the C Language Development 7
Utilities Option in Annex A, `ctags` produces unspecified results for C-language 7
source code files. It should write to standard error a message identifying this con- 7
dition and cause a nonzero exit status to be produced. 7

If the operand identifies FORTRAN source, the `ctags` utility shall produce an output line for each function definition. It may also produce output for any of the following objects:

— Subroutine definitions

— COMMON statements

— PARAMETER statements

— DATA and BLOCK DATA statements

— Statement numbers

On systems that do not support the FORTRAN Development Utilities Option in 7
Annex C, `ctags` produces unspecified results for FORTRAN source code files. It 7
should write to standard error a message identifying this condition and cause a 7
nonzero exit status to be produced. 7

It is implementation defined what other objects (including duplicate identifiers) produce output.

5.7.8 Exit Status

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

0 Successful completion.

>0 An error occurred.

5.7.9 Consequences of Errors

Default.

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

Usage, Examples

The special treatment of the tag `main` makes the use of `ctags` practical in directories with more than one program.

History of Decisions Made

The option list was significantly reduced from that provided by historical implementations. The `−F` option was omitted as redundant, since it is the default. The `−B` option was omitted as being of very limited usefulness. The `−t` option was omitted since the recognition of typedefs is now required for C source files.

The `−u` option was omitted because the update function was judged to be not only inefficient but also rarely needed.

An earlier draft included a `−w` option to suppress warning diagnostics. Since the types of such diagnostics could not be described, the option was omitted as being not useful.

The output with `−x` is meant to be a simple index that can be written out as an off-line readable function index. If the input files to `ctags` (such as `.c` files) were
not created using the same locales as those in effect when `ctags −x` is run, 8
results might not be as expected. 8

The description of C-language processing says ‘‘shall attempt to’’ because the C 7
language can be greatly obfuscated, especially through the use of `#defines`, and 7
this utility would be of no use if the real C preprocessor were run to identify them. 7
The output from `ctags` may be fooled and incorrect for various constructs. 7

The text for **LC_CTYPE** about compatibility with the C locale acknowledges that 7
the C Standard {7} imposes requirements on the locale used to process C source. 7
This could easily be a superset of that known as ‘‘the C locale’’ by way of imple- 7
mentation extensions, or one of a few alternative locales for systems supporting 7
different code sets. No statement is made for FORTRAN because ISO 1539 {2} 7
(FORTRAN 77) does not (yet) define a similar locale concept. However, a general 7
rule in POSIX.2 is that any time locales do not match (preparing a file for one 7
locale and processing it in another), results are suspect. 7

The collation sequence of the tags file is not affected by **LC_COLLATE** because it 8
is typically not used by human readers, but only by programs such as `vi` to locate 8
the tag within the source files. Using the POSIX Locale eliminates some of the 8
problems of coordinating locales between the `ctags` file creator and the `vi` file 8
reader. 8

Historically, the tags file has been used only by `ex` and `vi`. However, the format of the tags file has been published to encourage other programs to use the tags in
new ways. The format allows either regular expressions or line numbers to find 7
the identifiers because the historical `vi` recognizes either. The `ctags` utility does 7
not produce the format using line numbers because it are not useful following any 7
source file changes that add or delete lines. 7

Historical implementations also understand the objects used by the languages Pascal and sometimes LISP, and understand the C source output by `lex` and `yacc`. The `ctags` utility is not required to accommodate these languages, although implementors are encouraged to do so.

The following historical option was not specified, as `vgrind` is not included in this standard.

`−v` If the `−v` flag is given, an index of the form expected by `vgrind` is produced on the standard output. This listing contains the function name, file name, and page number (assuming 64 line pages). Since the output will be sorted into lexicographic order, it may be desired to run the output through `sort −f`. Sample use:

```
ctags −v files | sort −f > index
vgrind −x index
```

5.8 `df` — Report free disk space

5.8.1 Synopsis

`df [−kP]` [*file* . . .]

5.8.2 Description

The `df` utility shall write the amount of available space for file systems on which the invoking user has appropriate read access. File systems shall be specified by the *file* operands; when none are specified, information shall be written for all file systems. The format of the default output from `df` is unspecified, but all space figures shall be reported in 512-byte units, unless the `−k` option is specified. This output shall contain at least the file system names and amount of available space on each of these file systems.

5.8.3 Options

The `df` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`−k` Use 1024-byte units, instead of the default 512-byte units, when writing space figures.

`−P` Produce output in the format described in 5.8.6.1.

5.8.4 Operands

The following operand shall be supported by the implementation:

file A pathname of a file within the hierarchy of the desired file system. If a file other than a FIFO , a regular file, or a directory is specified, the results are unspecified. Otherwise, `df` shall write the amount of free space in the file system containing the specified *file* operand.

5.8.5 External Influences

5.8.5.1 Standard Input

None.

5.8.5.2 Input Files

None.

5.8.5.3 Environment Variables

The following environment variables shall affect the execution of `df`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.8.5.4 Asynchronous Events

Default.

5.8.6 External Effects

5.8.6.1 Standard Output

When both the `−k` and `−P` options are specified, the following header line shall be written (in the POSIX Locale):

`"Filesystem 1024-blocks Used Available Capacity Mounted on\n"` 8

When the `−P` option is specified without the `−k` option, the following header line shall be written (in the POSIX Locale):

`"Filesystem 512-blocks Used Available Capacity Mounted on\n"` 8

The implementation may adjust the spacing of the header line and the individual data lines so that the information is presented in orderly columns.

The remaining output with `−P` shall consist of one line of information for each specified file system. These lines shall be formatted as follows:

`"%s %d %d %d %d%% %s\n"`, *<file system name>*, *<total space>*,
<space used>, *<space free>*, *<percentage used>*, *<file system root>*

In the following list, all quantities expressed in 512-byte units (1024-byte when `-k` is specified) shall be rounded up to the next higher unit. The fields are:

<*file system name*> The name of the file system, in an implementation-defined format.

<*total space*> The total size of the file system in 512-byte units. The exact meaning of this figure is implementation defined, but should include <*space used*>, <*space free*>, plus any space reserved by the system not normally available to a user.

<*space used*> The total amount of space allocated to existing files in the file system, in 512-byte units.

<*space free*> The total amount of space available within the file system for the creation of new files by unprivileged users, in 512-byte units. When this figure is less than or equal to zero, it shall not be possible to create any new files on the file system without first deleting others, unless the process has appropriate privileges. The figure written may be less than zero.

<*percentage used*> The percentage of the normally available space that is currently allocated to all files on the file system. This shall be calculated using the fraction

$$\langle \textit{space used} \rangle / (\langle \textit{space used} \rangle + \langle \textit{space free} \rangle)$$

expressed as a percentage. This percentage may be greater than 100 if <*space free*> is less than zero. The percentage value shall be expressed as a positive integer, with any fractional result causing it to be rounded to the next highest integer.

<*file system root*> The directory below which the file system hierarchy appears.

5.8.6.2 Standard Error

Used only for diagnostic messages.

5.8.6.3 Output Files

None.

5.8.7 Extended Description

None.

5.8.8 Exit Status

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

0 Successful completion.

>0 An error occurred.

5.8.9 Consequences of Errors

Default.

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

Usage, Examples

The following example writes ‘‘portable’’ information about the `/usr` file system:

```
df -P /usr
```

Assuming that `/usr/src` is part of the `/usr` file system, the following will do the same as the previous example:

```
df -P /usr/src
```

On most historical systems, the ‘‘name of the file system, in an implementation-defined format’’ will be the special file on which the file system is mounted.

History of Decisions Made

The behavior of `df` with the `−P` option is the default action of the 4.2BSD `df` utility. The uppercase `−P` was selected to avoid collision with a known industry extension using `−p`.

Existing `df` implementations vary considerably in their default output. It therefore was necessary to describe the default output in a loose manner to accommodate all known existing implementations, and to add a portable option (`−P`) to provide information in a portable format.

The use of 512-byte units was retained because it is historical practice and to maintain compatibility with `ls` and other utilities in the base standard and the UPE. This does not mandate that the file system itself be based on 512-byte blocks. The `−k` option was added as a compromise measure. It was agreed in the working group that 512 was the best default unit because of its complete historical consistency on System V (versus the mixed 512/1024 usage on BSD systems), and that a `−k` option to switch to 1024-byte units was a good compromise. Users who prefer the more logical 1K quantity can easily alias `df` to `df -k` without breaking many historical scripts relying on the 512-byte units.

It was suggested that `df` and the various related utilities be modified to access a **BLOCKSIZE** environment variable to achieve consistency and user acceptance. Since this is not existing practice on any system, it is left as an interesting area for system extensions and will be re-evaluated in a future revision if it is widely implemented.

5.9 du — Estimate file space usage

5.9.1 Synopsis

`du` [−a | −s] [−kx] [*file* . . .]

5.9.2 Description

The `du` utility, by default, shall write to standard output the size of the file space 7
allocated to, and the size of the file space allocated to each subdirectory of, the file 7
hierarchy rooted in each of the specified files. The size of the file space allocated 7
to a file of type directory shall be defined as the sum total of space allocated to all 7
files in the file hierarchy rooted in the directory plus the space allocated to the 7
directory itself. 7

When `du` cannot stat [see *stat* () in POSIX.1 {8} 5.6.2] files, or stat or read directories, it shall report an error condition and the final exit status shall be affected. Files with multiple links shall be counted and written for only one entry. The
directory entry that is selected in the report is unspecified. By default, file sizes 7
shall be written in 512-byte units, rounded up to the next 512-byte unit.

5.9.3 Options

The `du` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

−a In addition to the default output, report the size of each file not of 7
type directory in the file hierarchy rooted in the specified file. 7
Regardless of the presence of the −a option, nondirectories given as *file* operands shall always be listed.

−k Write the files sizes in units of 1024 bytes, rather than the default 512-byte units.

−s Instead of the default output, report only the total sum for each of 7
the specified files. 7

−x When evaluating file sizes, evaluate only those files that have the same device as the file specified by the *file* operand.

5.9.4 Operands

The following operand shall be supported by the implementation:

file The pathname of a file whose size is to be written. If no *file* is specified, the current directory shall be used.

5.9.5 External Influences

5.9.5.1 Standard Input

None.

5.9.5.2 Input Files

None.

5.9.5.3 Environment Variables

The following environment variables shall affect the execution of du:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.9.5.4 Asynchronous Events

Default.

5.9.6 External Effects

5.9.6.1 Standard Output

The output from `du` shall consist of the amount of the space allocated to a file and the name of the file, in the following format: 7

`"%d %s\n"`, *<size>*, *<pathname>*

5.9.6.2 Standard Error

Used only for diagnostic messages.

5.9.6.3 Output Files

None.

5.9.7 Extended Description

None.

5.9.8 Exit Status

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

0 Successful completion.

>0 An error occurred.

5.9.9 Consequences of Errors

Default.

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

Usage, Examples

The use of 512-byte units is historical practice and maintains compatibility with `ls` and other utilities in the base standard and the UPE. This does not mandate that the file system itself be based on 512-byte blocks. The `−k` option was added as a compromise measure. It was agreed in the working group that 512 was the best default unit because of its complete historical consistency on System V (versus the mixed 512/1024 usage on BSD systems), and that a `−k` option to switch to 1024-byte units was a good compromise. Users who prefer the 1K quantity can easily alias `du` to `du -k` without breaking the many historical scripts relying on the 512-byte units.

History of Decisions Made

The `-b` option was added to an earlier draft to provide a resolution to the situation where System V and BSD systems give figures for file sizes in *blocks*, which is an implementation-defined concept. (In common usage the block size is 512 bytes for System V and 1024 bytes for BSD systems.) However, `-b` was later deleted, since the default was eventually decided as 512-byte units.

Historical file systems provided no way to obtain exact figures for the space allocation given to files. There are two known areas of inaccuracies in traditional file systems: cases of *indirect blocks* being used by the file system or *sparse* files yielding incorrectly high values. An indirect block is space used by the file system in the storage of the file, but that need not be counted in the space allocated to the file. A *sparse* file is one in which an *lseek*() (see 6.5.3 in POSIX.1 {8})call has been made to a position beyond the end of the file and data has subsequently been written at that point. A file system need not allocate all the intervening zero-filled blocks to such a file. It is up to the implementation to define exactly how accurate its methods are.

The `-a` and `-s` options were mutually exclusive in the original version of `du`. The 7
POSIX.2 description is implied by the language in the *SVID*, where `-s` is described 7
as causing ‘‘only the grand total’’ to be reported. Some systems may produce out- 7
put for `-sa`, but a strictly conforming POSIX.2 application cannot use that combi- 7
nation. 7

The `-a` and `-s` options were adopted from the *SVID*, except that System V's behavior of not listing nondirectories explicitly given as operands, unless the `-a` option is specified, was considered a bug; the BSD-based behavior (report for all operands) is mandated. The default behavior of `du` in the *SVID* with regard to reporting the failure to read files (it produces no messages) was considered counter-intuitive, and thus it was specified that the POSIX.2 default behavior shall be to produce such messages. These messages can be turned off with shell redirection to achieve the System V behavior.

The `-x` option is existing practice on recent BSD systems. It has been adopted by POSIX.2 because there was no other historical method of limiting the `du` search to a single file hierarchy. This limitation of the search is necessary to make it possible to obtain file space usage information about a file system on which other file systems are mounted, without having to resort to a lengthy `find` and `awk` script.

5.10 ex — Text editor

5.10.1 Synopsis

`ex` [−rR][−s | −v][−c *command*][−t *tagstring*][−w *size*][*file* ...]

Obsolescent Version:

`ex` [−rR][− | −v][+*command*][−t *tagstring*][−w *size*][*file* ...] 8

5.10.2 Description

The `ex` utility is a line-oriented text editor that shall support both line and full-screen editing (see `vi` in 5.35).

Certain block-mode terminals do not have all the capabilities necessary to support the complete `ex` definition, such as the full-screen editing commands (*visual* 7 mode) or *open mode*. When these commands cannot be supported on such termi- 7 nals, this condition shall not produce an error message such as ''not an editor command'' nor report a syntax error. The implementation may either accept the commands and produce results on the screen that are the result of an unsuccessful attempt to meet the requirements of this standard or report an error describing the terminal-related deficiency. The affected commands are noted as they occur later in this clause.

5.10.3 Options

The `ex` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

−c *command*
+*command* (Obsolescent.)
: Begin editing by executing the specified `ex` command-mode command(s). As with normal editing command-line entries, the *command* option-argument can consist of multiple `ex` commands separated by vertical-line characters (|). The use of commands that enter input or visual modes in this manner produces undefined results.

−r
: Recover the named files after an editor or system crash, after the editor has been terminated by a signal, or after the use of a `preserve` editor command. A ''crash'' in this context is an unexpected failure of the system or utility that requires restarting the failed system or utility. A system crash implies that any utilities running at the time also crash. In the case of an editor or system crash, the degree of recovery (the number of changes to the buffer since the most recent `preserve` command) available is

unspecified.

If no *file* operands are given, all other options, the **EXINIT** variable, and any `.exrc` files shall be ignored; a list of all recoverable files available to the invoking user shall be written; and `ex` shall exit without reading files or processing user commands.

`−R` Set read-only mode, preventing accidental overwriting of the files. Any command that would write to a file shall require the ! suffix (see, for example, 5.10.7.2.37) to be effective in this mode.

`−s`
`-` (Obsolescent.)
Prepare `ex` for batch use by taking the following actions:

— Suppress writing prompts and informational (but not diagnostic) messages.

— Ignore the value of **TERM** and any implementation default terminal type and assume the terminal is a type incapable of supporting visual mode; see 5.10.7.2.36 and the description of `vi` in 5.35.

— Suppress the use of the **EXINIT** environment variable (see 5.10.5.3) and the reading of any `.exrc` file (see 5.10.7).

`−t` *tagstring*
Edit the file containing the specified *tagstring* and proceed as if the first command were `:tag` *tagstring* . (See `ctags` in 5.7). The tags feature represented by `−t` *tagstring* and the `ta` command (see 5.10.7.2.32) is optional. It shall be provided on any system that also provides a conforming implementation of `ctags`; otherwise, the use of `−t` produces undefined results.

`−v` Invoke `vi` (see 5.35).

`−w` *size* Set the value of the *window* editor option to *size* .

If both the `−t` *tagstring* and `−c` *command* (or the obsolescent +*command*) options are given, the `−t` *tagstring* shall be processed first; i.e., the file containing the tag is selected by `−t` and then the command is executed.

5.10.4 Operands

The following operand shall be supported by the implementation:

file A pathname of a file to be edited.

5.10.5 External Influences

5.10.5.1 Standard Input

The standard input shall be a text file consisting of commands, as described in 5.10.7.

5.10.5.2 Input Files

Input files shall be text files or files that would be text files except for an incom- 7
plete last line that is not longer than {LINE_MAX} − 1 bytes in length and contains 7
no NUL characters. The editing of other forms of files may optionally be allowed 7
by `ex` implementations. 7

The `.exrc` files (see 5.10.7) shall be text files consisting of commands. 8

By default, `ex` shall read lines from the files to be edited without interpreting any 8
of those lines as any form of editor command. 8

5.10.5.3 Environment Variables

The following environment variables shall affect the execution of `ex`:

COLUMNS This variable shall override the system-selected horizontal 7
screen size. See 2.6 for valid values and results when it is 7
unset or null. 7

EXINIT This environment variable shall be interpreted to contain a list of `ex` commands that are executed on editor startup, before reading the first file. The list can contain multiple commands by separating them using a vertical-line (`|`) character. See 5.10.7 for more details of the initialization phase.

HOME This variable shall be interpreted as a pathname of a directory that shall be searched for an editor startup file named `.exrc`; see 5.10.7 for details.

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_COLLATE This variable shall determine the locale for character collation information in regular expressions.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files), the behavior of character classes within regular expressions, the classification of characters as upper- or lowercase letters, the case conversion of letters, and the detection of word boundaries.

LC_MESSAGES This variable shall determine the processing of affirmative 7
responses and the language in which messages should be 7
written. 7

LINES This variable shall override the system-selected vertical 7
screen size, used as the number of lines in a screenful and 7
the vertical screen size in visual mode. See 2.6 for valid 7
values and results when it is unset or null. 7

PATH This variable shall determine the search path for the shell 8
command specified in the editor commands `shell`, `read`,
and `write`; see the description of command search and
execution in 3.9.1.1.

SHELL This variable shall be interpreted as the preferred command-line interpreter for use in `!`, `shell`, `read`, and other commands with an operand of the form `!`*string* . For the `shell` command, the program shall be invoked with the single argument `-i`, for all others it shall be invoked with the two arguments `-c` and *string* . If no **SHELL** environment variable is set, or it is set to a null string, the `sh` (see 4.56) utility shall be used.

TERM This variable shall be interpreted as the name of the ter-
minal type. If this variable is unset or null, an 7
unspecified default terminal type shall be used. 7

5.10.5.4 Asynchronous Events

The following actions shall be taken upon receipt of signals:

SIGINT When an interrupt occurs, `ex` shall alert the terminal and write a message. The current editor command shall be aborted and `ex` shall return to the command level and prompt for another command. If the standard input is not a terminal device, `ex` shall exit at the interrupt and return a nonzero exit status. (The alerting action can be modified by the use of the `errorbells` editor option; see 5.10.7.5.)

SIGCONT The screen shall be refreshed (if in visual mode).

SIGHUP If the current buffer has changed since the last `e` or `w` command, `ex` shall attempt to save the current file in a state such that it can be recovered later by an `ex -r` command. 8

The action taken for all other signals is unspecified. 7

5.10.6 External Effects

5.10.6.1 Standard Output

The standard output shall be used only for writing prompts to the user, for informational messages, and for writing lines from the file.

5.10.6.2 Standard Error

Used only for diagnostic messages.

5.10.6.3 Output Files

The output from `ex` shall be text files that are identical to the input files if no changes have been made to the files by commands, with the exception that in all cases where a forced session termination (the `ex` command `q!`; see 5.10.7.2.23) has not been issued prior to any file write, a trailing `<newline>` shall be added to the last line of the file if one was not present in the input.

5.10.7 Extended Description

The pathname of the file being edited by `ex` is referred to as the *current* file. The text of the file shall be read into a working version of the file (called *buffer* in this clause, although no attempt is made to imply a specific implementation) and all editing changes shall be performed on that version; the changes shall have no effect on the original file until an `ex` command causes the file to be written out. Lines in the buffer may each be limited to {LINE_MAX} bytes and an error message may be written if the limit is exceeded during editing.

The *alternate* pathname is the name of the last file mentioned in an editor command, or the previous current pathname if the last file mentioned became the current file. When the character `%` appears in a pathname entered as part of a command argument, it shall be replaced by the current pathname; the character `#` shall be replaced by the alternate pathname. Any character, including `%` and `#`, shall retain its literal value (be escaped) when preceded by a backslash.

When an error occurs, `ex` shall alert the terminal and write a message. (The alerting action can be modified by the use of the `errorbells` editor option; see 5.10.7.5.)

If the system crashes, `ex` shall attempt to preserve the buffer if any unwritten changes were made. The command-line option `-r` can be used to retrieve the saved changes.

During initialization, before the first file is read or any user commands from the terminal are processed, if the environment variable **EXINIT** is set, the editor shall execute the `ex` commands contained in that variable. If the variable is not set, `ex` shall attempt to read commands from the file `$HOME/.exrc` (the file `.exrc` in the directory referred to by the **HOME** environment variable). If and only if **EXINIT** or `$HOME/.exrc` sets the editor option `exrc`, `ex` finally shall attempt to read commands from a file `.exrc` in the current directory. In the event that **EXINIT** is not set and the current directory is the user's home directory, any `.exrc` file shall only be processed once. No `.exrc` file shall be read unless it is owned by the same user ID as the effective user ID of the process. After any `.exrc` files are processed, any commands specified by the `−c` option shall be processed.

By default `ex` shall start in the command mode, which shall be indicated by the `":"` prompt. The input mode can be entered by `append`, `insert`, or `change` commands; it can be exited (and command mode re-entered) by typing a period (`.`) alone at the beginning of a line. There is one other mode, visual mode, in which full-screen editing is available. This is described more fully under the `visual` command and in 5.35. The command line can consist of multiple `ex` commands separated by vertical-line characters (`|`). The use of commands that enter input or visual modes in this manner, unless they are the final command on the line, produces undefined results.

Command lines beginning with the double-quote character (`"`) shall be ignored. This can be used for comments in an editor script.

5.10.7.1 Addressing

Addressing in `ex` relates to the *current line*. In general, the current line shall be the last line affected by a command; the exact effect on the current line is discussed under the description of each command. When the buffer contains no lines, the current line shall be set to zero.

Addresses shall be constructed by one of the following methods:

(1) The address `.` (period) refers to the current line.

(2) The address `$` refers to the last line of the buffer.

(3) The address *n*, where *n* shall be a decimal number, refers to the *n*-th line of the buffer.

(4) The address *'x* refers to the line marked with the mark-name character *x*, which shall be a lowercase letter of the POSIX Locale. Lines can be marked with the `ma` or `k` commands described below.

(5) A regular expression (RE) enclosed by slashes (`/`) is an address, and refers to the first line found by searching forward from the line following the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. The second slash can be omitted at the end of a command line. If the `wrapscan` option is set, the search shall wrap around to the beginning of the buffer and continue up to and including the current line, so that the entire buffer is searched. (See

5.10.7.3.)

(6) A RE enclosed in question marks (`?`) addresses the first line found by searching backward from the line preceding the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. The second question mark can be omitted at the end of a command line. If the `wrapscan` option is set, the search shall wrap around from the beginning of the buffer to the end of the buffer and continue up to and including the current line, so that the entire buffer is searched. (See 5.10.7.3.)

(7) An address followed by a plus sign (`+`) or a minus sign (`–`) followed by a decimal number is an offset address, and refers to the first address plus (respectively minus) the indicated number of lines. If the address is omitted, the addition or subtraction shall be taken with respect to the current line.

(8) An address of `+` or `–` followed by a number shall be taken with respect to the current line number; for example, `–5` is understood to mean `.–5`.

(9) An address ending with `+` or `–` shall have 1 added to or subtracted from the address, respectively. As a consequence of this rule and of rule (8) above, the address `–` refers to the line preceding the current line. Moreover, trailing `+` and `–` characters have a cumulative effect; for example, `––` refers to the current line less 2.

(10) A percent sign (`%`) shall stand for the address pair `1,$`.

Commands require zero, one, or two addresses. See the descriptions of *line* and *range* in 5.10.7.2. Commands that require zero addresses shall regard the presence of an address as an error. 7

Adjacent addresses in a *range* shall be separated from each other by a comma (`,`) or a semicolon (`;`). In the latter case, the current line (`.`) shall be set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches [see rules (5) and (6) above]. The second address of any two-address sequence shall correspond to a line that follows, in the buffer, the line corresponding to the first address. The first address shall be less than or equal to the second address. The first address shall be greater than or equal to the first line of the editing buffer and the last address shall be less than or equal to the last line of the editing buffer. Any other case shall be an error. 7

5.10.7.2 Command Descriptions

The following symbols are used in this subclause to represent optional modifiers. Any or all can be omitted; the defaults are shown.

line A single line address, given in any of the forms described in 5.10.7.1; the default for *line* shall be the current line.

range A *line* , or a pair of line addresses, separated by a comma or semicolon (see 5.10.7.1 for the difference between the two); the default for *range* shall be the current line only (`.,.`). A percent sign (%) shall stand for the range `(1,$)`. If the range specified is such that the starting address exceeds the ending address, the range is invalid and the command shall not be performed. If more than the expected number of addresses are given in a range, the greatest valid number of the last one(s) given shall be used. For example, `1,3,5p` prints lines 3 through 5 (because two is the greatest valid number in the range accepted by `print`).

count A positive integer, specifying the number of lines to be affected by the command; the default for *count* shall be 1.

flags One of the characters `#`, `p`, or `l` (ell), or both `#` and `l` to add numbers to list-format output. When a command with such a flag completes, the addressed line(s) shall be written out as if by the corresponding `#`, `p`, or `l` command. The use of *flags* shall apply to all lines written by the `list`, `number`, `open`, `print`, `substitute`, `visual`, &, and `z` commands; for other commands, it shall apply to the current line at the completion of the command. In addition, any number of + or – characters can also be given after the flags, in which case the line written shall not be the one affected by the command, but rather the line addressed by the offset address as described above. The default for *flags* shall be null.

buffer One of a number of named areas for saving text. The named buffers are specified by the lowercase letters of the POSIX Locale. Specifying *buffer* shall cause the area of text affected by the command to be stored into the buffer as it was before the command took effect. This argument is also used on the `put` command and the visual mode ''put'' commands (`p` and `P`) to specify the buffer that shall provide the text to insert.

If the buffer name is specified in uppercase, and the buffer is to be modified (as with a deletion or yanking command), the buffer shall be appended to rather than being overwritten. If the buffer is not to be modified (as in a visual mode ''put'' commands) the buffer name can be specified in lowercase or uppercase with the same results. There shall be also one unnamed buffer, which is the repository for all text deleted (with the `delete` or visual mode `d` command) or yanked (with the `yank` or visual mode `y` command) when no buffer is specified.

There are also numbered buffers, 1 through 9, which shall be accessible only from visual mode. These buffers are special in that, in visual mode, when deleted text is placed in the unnamed buffer, it also shall be placed in buffer 1, the previous contents of buffer 1 shall be placed in buffer 2, and so on. Any text in buffer 9 shall be lost. Text that is yanked (or otherwise copied) into the unnamed buffer shall not modify the numbered buffers. Text cannot be placed directly into the numbered buffers although it can be retrieved from them by using a visual mode ''put'' command with the buffer name given as a number. When the *buffer* modifier is not used in the commands below, the unnamed buffer shall be the default.

file A pattern used to derive a pathname; the default shall be the current file, as defined above. If no current file has yet been established, a warning shall be written and the command shall be aborted, except where specifically noted in the individual command descriptions that follow. The pattern shall be subjected to the process of shell word expansions (see 3.6); if more than a single pathname results and the command is expecting one file, the effects are unspecified.

word In the POSIX Locale, a *word* consists of a maximal sequence of letters, digits, and underscores, delimited at both ends by characters other than letters, digits, or underscores, or by the beginning or end of a line or the file.

! A character that can be appended to the command to modify its operation, as detailed in the individual command descriptions.

If both a *count* and a *range* are specified for a command that uses them, the number of lines affected shall be taken from the *count* value rather than the *range*. The starting line for the command shall be taken to be the first line addressed by the *range*.

When only a *line* or *range* is specified with no command, the implied command shall be either a `print`, `list`, or `number` (`p`, `l`, or `#`). The command selected shall be the last of these three commands to be used, including use as a *flag*. When no range or count is specified and the command line is a blank line, the current line shall be written, and the current line shall be set to `.+1`.

Zero or more `<blank>` characters can precede or follow the addresses, *count*, *flags*, or command name. Any object following a command name (such as *buffer*, *file*, etc.) that begins with an alphabetic character shall be separated from the command name with at least one `<blank>`.

For each of the commands listed below, the command can be entered as the abbreviation (those characters in the Synopsis command word preceding the [), the full command (all characters shown for the command word, omitting the [and]), or any subset of the characters of the full command down to the abbreviation. For example, the `args` command (shown as `ar[gs]` in the Synopsis) can be entered as `ar`, `arg`, or `args`.

5.10.7.2.1 abbrev

Synopsis : `ab[brev]` *word rhs*

Add the named abbreviation to the current abbreviation list. In visual mode, if *word* is typed so that it is preceded and followed by characters that cannot be part of a *word* (as defined previously), it shall be replaced by the string *rhs*.

5.10.7.2.2 append

Synopsis : [*line*] `a[ppend][!]`

Enter input mode; the input text shall be placed after the specified line. If line 0
is specified, the text shall be placed at the beginning of the buffer. The current 7
line indicator shall be set to the last input line; if no lines are input, it shall be set 7
to the target line, or to the first line of the file if a target of 0 was specified. Fol- 7
lowing the command name with ! shall cause the `autoindent` editor option set-
ting to be toggled for the duration of this command only.

5.10.7.2.3 args

Synopsis : `ar[gs]`

Write the argument list with the current argument inside `[` and `]`. The argument list is the list of operands on startup, which can subsequently be replaced by the operands of the `next` command.

5.10.7.2.4 change

Synopsis : [*range*] `c[hange][!]` [*count*]

Enter input mode; the input text shall replace the specified lines (*range*). The 7
current line indicator shall be set to the last line input; if no lines were input, it 7
shall be set to the line before the target line, or to the first line of the file if there 7
are no lines preceding the target. Following the command name with ! shall 7
cause the `autoindent` editor option setting to be toggled for the duration of this 8
command only. 8

5.10.7.2.5 chdir

Synopsis : `chd[ir][!]` [*directory*]
Synopsis : `cd[!]` [*directory*]

Change the current working directory to *directory*. If the current buffer has been modified since the last write, a warning shall be written and the command shall fail. This warning can be overridden by appending ! to the command name, which shall allow the command to complete.

5.10.7.2.6 copy

Synopsis : [*range*] `co[py]` *line* [*flags*]
Synopsis : [*range*] `t` *line* [*flags*]

Place a copy of the specified lines (*range*) after the specified destination *line*; line 0 specifies that the lines shall be placed at the beginning of the buffer.

5.10.7.2.7 **`delete`**

Synopsis : [*range*] `d[elete]` [*buffer*] [*count*] [*flags*]

Delete the specified lines from the buffer. If a named *buffer* is specified, the deleted text shall be saved in it; otherwise, the deleted text shall be saved in the unnamed buffer. If the command name is followed by a letter that could be interpreted as either a buffer name or a *flag* value (because neither a *count* nor an additional *flags* value was given), `ex` shall consider the letter to be a *flags* value if the letter directly follows the command name, without any `<blank>` separation; if the letter is preceded by `<blank>`(s), it shall be considered a buffer name.

For example:

`1dp` or `1deletep`	Deletes the first line and prints the line that was second.
`1d p`	Deletes the first line, saving it in buffer `p`.
`1d p1l`	(Pee-one-ell.) Deletes the first line, saving it in buffer `p`, and listing the line that was second.

The current line indicator shall be set to the line following the deleted lines, or to the last line if the deleted lines were at the end.

5.10.7.2.8 **`edit`**

Synopsis : `e[dit][!]` [`+`*line*] [*file*]
Synopsis : `ex[!]` [`+`*line*] [*file*]

Begin editing *file*. If the current buffer has been modified since the last write, a warning shall be written and the command shall be aborted. This action can be overridden by appending the character `!` to the command name (`e!` *file*). The current line indicator shall be the last line of the buffer; however, if this command is executed from within visual mode, the current line indicator shall be the first line of the buffer. If the `+`*line* option is specified, the current line indicator shall be set to the specified position, where *line* can be a number (or `$`) or specified as "`/`*pattern* " or "`?`*pattern* ". Preceding the pattern with a `/` shall start a search from the beginning of the file. Preceding the pattern with a `?` shall start a search from the end of the file. This command shall be affected by the editor options `autowrite` and `writeany`; see 5.10.7.5 for details.

5.10.7.2.9 **`file`**

Synopsis : `f[ile]` [*file*]

Write the current pathname, the number of lines, the current position when no *file* argument has been specified; `ex` may write other unspecified information. If no current file has yet been established, an unspecified message shall be written to indicate that no file is being edited. With *file*, `ex` shall change the current file name to *file*, without changing the contents of the buffer or the previous current file.

5.10.7.2.10 `global`

Synopsis : [*range*] `g[lobal]` /*pattern* / [*cmds*] 7
Synopsis : [*range*] `v` /*pattern* / [*cmds*] 7

Mark the lines within the given range that match (`g`) or do not match (`v`) the given pattern. Then execute the `ex` command(s) given by *cmds* with the current line (`.`) set to each marked line.

Multiple *cmds* can be specified, one per line, by escaping each `<newline>` with a backslash. If *cmds* are omitted, each line shall be written. For the `append`, `change`, and `insert` commands, the input text shall be included as part of the `global` command line; in this case the terminating period can be omitted if it ends *cmds* . The `visual` command can be specified as one of the *cmds* . In this mode, input shall be taken from the terminal. Entering a `Q` from visual mode shall cause the next line matching the pattern to be selected and visual mode to be reentered, until the list is exhausted.

The `global` command itself and the `undo` command cannot be used in *cmds* . The editor options `autoprint`, `autoindent`, and `report` shall be inhibited for the duration of the `g` or `v` command.

5.10.7.2.11 `insert`

Synopsis : [*line*] `i[nsert][!]`

Enter input mode; the input text shall be placed before the specified line. The 7
current line indicator shall be set to the last line input; if no lines were input, it 7
shall be set to the line before the target line, or to the first line of the file if there 7
are no lines preceding the target. Following the command name with the charac- 7
ter ! shall cause the `autoindent` editor option setting to be toggled for the dura- 8
tion of this command only. 8

5.10.7.2.12 `join`

Synopsis : [*range*] `j[oin][!]` [*count*] [*flags*]

Join the text from the specified lines together into one line. In the POSIX Locale, when the last character on the first line of a pair of lines to be joined is a period, two `<space>`s shall be added following the period; when the last character of the first line is a `<blank>` or when the first character on the second line of the pair is a `)`, no `<space>`s shall be added; otherwise, one `<space>` shall be added following the last character of the first line. Extra `<blank>`s at the start of a line shall be discarded.

Appending a character ! to the `join` command name shall cause a simpler join with no white-space processing, independent of the current locale.

5.10.7.2.13 `list`

Synopsis : [*range*] `l[ist]` [*count*] [*flags*]

Write the addressed lines in a way that should be unambiguous: nonprintable
characters shall be written as implementation-defined multicharacter sequences; 7
the end of the line shall be marked with a `$`.

Long lines shall be folded; the length at which folding occurs is unspecified, but should be appropriate for the output device. The only useful flag is #, for line numbers. The current line indicator shall be set to the last line written.

5.10.7.2.14 `map`

Synopsis : `map[!]` [*x rhs*]

Define macros for use in visual mode. The first argument shall be a single character. When this character is typed in visual mode, the action shall be as if the corresponding *rhs* had been typed. If the character ! is appended to the command name `map`, the mapping shall be effective during input mode rather than command
mode. This allows *x* to have two different macro definitions at the same 7
time: one for command mode and one for input mode. Nonprintable characters, 7
except for `<tab>`, require escaping with a `control-V` (or `control-Q`) to be 8
entered in the arguments. On certain block-mode terminals, the mapping need not occur immediately (for example, it may occur after the terminal transmits a group of characters to the system), but it shall achieve the same results of modifying the file as if it occurred immediately. Implementations may restrict the set of commands accepted within *rhs* ; the list of restrictions is implementation defined.

The `map` command with no arguments shall write all of the macros currently defined. If ! is appended to the command, only the macros effective during input mode shall be written; otherwise, only the macros effective during command mode shall be written.

5.10.7.2.15 `mark`

Synopsis : [*line*] `ma[rk]` *x*
Synopsis : [*line*] `k` *x*

Give the specified line the specified mark *x*, which shall be a single lowercase letter of the POSIX Locale. The current line position shall not be affected. The expression `'`*x* can then be used as an address in any command requiring one. For example ‘‘`.,'`*x*`d`’’ shall delete all the lines from the current one to the marked line. Also see 5.35.7.1.23 and 5.35.7.1.24 for uses of the mark in visual mode. If the `'`*x* command is used in nonvisual mode, the character marked shall be the first `<non->`blank character of the current line. Otherwise, the character marked shall be the character at the current column of the current line.

5.10.7.2.16 move

Synopsis : [*range*] m[ove] *line*

Move the specified lines (*range*) to be after the target line (*line*). The current line indicator shall be set to the first of the moved lines.

5.10.7.2.17 next

Synopsis : n[ext][!] [*file* ...]

Edit the next file from the argument list. If the current buffer has been modified since the last write, a warning shall be written and the command shall be aborted. This action can be overridden by appending the character ! to the command name (n!). The argument list can be replaced by specifying a new one as operands to this command. Editing then starts with the first file on this new list. The current line indicator shall be reset as described for the edit command. This command shall be affected by the editor options autowrite and writeany; see 5.10.7.5 for details.

5.10.7.2.18 number

Synopsis : [*range*] nu[mber] [*count*] [*flags*]
Synopsis : [*range*] # [*count*] [*flags*]

Write the selected lines, each preceded with its line number in decimal. Non-printable characters, except for <tab>, shall be expanded as specified by the print command. See 5.10.7.2.21. The format shall be as follows:

"%+6d\t%s", *<line number>* , *<line text>*

The only meaningful flag is l, which shall cause the additional expanded writing of <tab>s and end-of-lines done by the list command to be performed. The current line indicator shall be set to the last line written.

5.10.7.2.19 open

Synopsis : [*line*] o[pen] /*pattern* / [*flags*]

Enter open mode, which shall be equivalent to visual mode with a one-line window. All the visual mode commands shall be available. If a match is found for the optional regular expression in *line* , the cursor shall be placed at the start of the matching pattern. The visual mode command Q (see 5.35.7.1.71) shall exit open mode. This command need not be supported on block-mode terminals.

5.10.7.2.20 preserve

Synopsis : pre[serve]

Save the current buffer in a form that can later be recovered by using ex −r or by using the recover command. After the file has been preserved, a mail message shall be sent to the user. This message shall be readable by invoking the mailx utility (see 4.40). The message shall contain the name of the file, the time of preservation, and an `ex` command that could be used to recover the file. Additional information may be included in the mail message.

5.10.7.2.21 **print**

Synopsis : [*range*] `p[rint]` [*count*] [*flags*]

Write the addressed lines. Nonprintable characters, except for `<tab>`, shall be written as implementation-defined multicharacter sequences. 7

Long lines shall be folded; the length at which folding occurs is unspecified, but should be appropriate for the output device. The only meaningful *flags* are # and `l`. The current line indicator shall be set to the last line written.

5.10.7.2.22 **put**

Synopsis : [*line*] `pu[t]` [*buffer*]

Put back deleted or ''yanked'' lines after line *line* . A buffer can be specified; otherwise, the text in the unnamed buffer (where deleted or yanked text is placed by default) shall be restored. The current line indicator shall be set to the first line put back.

5.10.7.2.23 **quit**

Synopsis : `q[uit][!]`

Terminate the editing session. If the current buffer has been modified since the last write, a warning shall be written and the command shall fail. This warning can be overridden, and an exit forced, discarding changes, by appending the character ! to the command name.

5.10.7.2.24 **read**

Synopsis : [*line*] `r[ead][!]` [*file*]

Place a copy of the specified file in the current buffer after the target line (which can be line 0 to place text at the beginning). If no *file* is named, the current file shall be the default. If there is no current file, then *file* shall become the current file. If there is no current file nor *file* operand, the command shall fail.

The current line indicator shall be set to the last line read. In visual mode, the current line indicator shall be set to the first line read. If *file* is preceded by `!`, *file* shall be taken to be an operating system command and passed to the program named in the **SHELL** environment variable; the resultant output shall be read in to the buffer. The special meaning of ! can be overridden by escaping it with a backslash character.

5.10.7.2.25 **recover**

Synopsis : `rec[over]` *file*

Attempt to recover *file* if it was saved as the result of a `preserve` command, the receipt of a signal (see 5.10.5.4), or a system or editor crash. The current line indicator shall be reset as described for the `read` editor command.

5.10.7.2.26 `rewind`

Synopsis : `rew[ind][!]`

Rewind the argument list; that is, the current file shall be set to the first file in the argument list. This shall be equivalent to a `next` command with the current argument list as its operands. If the current buffer has been modified since the last write, a warning shall be written and the command shall be aborted. The action can be overridden by appending the character ! to the command name (`rew!`). The current line indicator shall be reset as described for the `read` editor command. This command shall be affected by the editor options `autowrite` and `writeany`; see 5.10.7.5 for details.

5.10.7.2.27 `set`

Synopsis : `se[t]` [*option* [=[*value*]] . . .] [`no`*option* . . .] [*option* `?` . . .] [`all`] 7

When no arguments are specified, write those options whose values have been 7
changed from the default settings; when the argument `all` is specified, write all 7
of the option values. 7

Giving an option name followed by the character `?` shall cause the current value of that option to be written. The `?` can be separated from the option name by zero or more `<blank>`s. The `?` shall be necessary only for Boolean valued options. Boolean options can be given values by the form `se` *option* to turn them on or `se no`*option* to turn them off; string and numeric options can be assigned by the form `se` *option* =*value* . Any `<blank>`s in strings can be included as is by preceding
each such character with a backslash. More than one option can be set or listed 7
by a single `set` command by specifying multiple arguments, each separated from 7
the next by one or more `<blank>`s.

See 5.10.7.5 for further details about options.

5.10.7.2.28 `shell`

Synopsis : `sh[ell]`

Invoke the program named in the **SHELL** environment variable with the argument `−i` (interactive mode). Editing shall be resumed when the program exits.

5.10.7.2.29 `source`

Synopsis : `so[urce]` *file*

Read and execute commands from the file specified by the mandatory argument *file*. Such `so` commands can be nested. The maximum supported nesting depth is implementation defined, but shall be at least one.

5.10.7.2.30 `substitute`

Synopsis : [*range*] `s[ubstitute]` [/*pattern* /*repl* /[*options*] [*count*] [*flags*]] 7

Replace the first instance of the pattern *pattern* by the string *repl* on each specified line. (See 5.10.7.3 and 5.10.7.4.) If the /*pattern* /*repl* / argument is not present, the /*pattern* /*repl* / from the previous `substitute` command shall be used. If *options* includes the letter `g` (global), all nonoverlapping instances of the pattern in the line shall be substituted. If the option letter `c` (confirm) is included, then before each substitution the line shall be written with ^ characters written on the following line, adjacent to and identifying the pattern to be replaced; an affirmative response shall cause the substitution to be done, while any other input aborts it. An affirmative response shall consist of a line with the affirmative response (as defined by the current locale) at the beginning of the line. Such a line shall be subject to editing in the same way as the command line (the `/` or `:` line at the bottom of the screen). The current line indicator shall be set to the last line substituted. When the `c` option is used, typing the interrupt character or receiving the SIGINT signal shall stop the substitute operation and `ex` shall return to command mode. All substitutions completed before the interrupt occurred shall be retained, and none shall be made after that point. The current line indicator shall be set to the last line substituted. This command shall be affected by the **LC_MESSAGES** environment variable and the `wrapscan` option.

5.10.7.2.31 `suspend`

Synopsis : `su[spend][!]`
Synopsis : `st[op][!]`

Allow control to return to the invoking process; `ex` shall suspend itself as if it had received the SIGTSTP signal. The suspension shall occur only if the system supports job control as described in POSIX.1 {8} and job control is enabled in the invoking shell (see the description of `set −m` in 3.14.11).

If job control is not supported or is disabled in the invoking shell, issuing either the `suspend` or `stop` command shall produce a warning message, but otherwise shall have no effect. Following either `suspend` or `stop` with the character `!` shall affect the operation of the `autowrite` editor option for this command only.

The current *susp* character (see `stty` in 4.59) shall also cause the suspension.

5.10.7.2.32 `tag`

Synopsis : `ta[g][!]` *tagstring*

Search for the *tagstring* , which can be in a different file. If the tag is in a different file, then the new file shall be opened for editing. If the current buffer has been modified since the last write, a warning shall be written and the command shall be aborted. The action can be overridden by appending the character `!` to the command name. The current line indicator shall be reset to the line indicated by the tag. This command shall be affected by the editor options `autowrite` and `writeany`; see 5.10.7.5 and 5.7 (`ctags`) for details. This command shall be affected by the `tags` editor option.

The `tag` command shall search for *tagstring* in the tag file referred to by the `tags`
editor option until a reference to *tagstring* is found. The file pointed to by this
reference shall be loaded into the buffer, and the current line shall be set to the
first occurrence of the pattern specified in the tags file associated with the sup- 7
plied *tagstring* ; if the tags file contained a line-number reference, the current line 7
shall be set to that line. If the pattern or line number is not found, an error mes- 7
sage shall be written. If a file referred to by the `tags` editor option does not exist
or is not readable, an error message shall be written. The results are unspecified 8
if the format of a tags file is not as specified by the `ctags` utility description. 8

5.10.7.2.33 unabbrev

Synopsis : `una[bbrev]` *word*

Delete *word* from the list of abbreviations, as described by the `abbrev` editor command in 5.10.7.2.1.

5.10.7.2.34 undo

Synopsis : `u[ndo]`

Reverse the changes made by the previous editing command (one that changes the contents of the buffer). For this purpose, `global` and `visual` are considered single commands. An `undo` can itself be reversed. Commands that affect the external environment, such as `write`, `edit`, and `next` cannot be undone.

5.10.7.2.35 unmap

Synopsis : `unm[ap][!]` *x* 7

If no ! is specified, remove the command-mode macro definition for *x*; otherwise, 7
remove the input-mode macro definition for *x*. See 5.10.7.2.14. 7

5.10.7.2.36 visual

Synopsis : [*line*] `vi[sual]` [*type*] [*count*] [*flags*]

Enter visual mode with the current line indicator set to *line* . The *type* is optional,
and can be −, `.`, `+`, or `^`, as in the `z` command, to specify the position of the
specified line on the screen window. (The default shall be to place the line at the
top of the screen window.) A *count* specifies the number of lines that will initially
be written; the default shall be the value of the editor option `window`. The com-
mand `Q` shall exit visual mode. (For more information, see `vi` in 5.35). This com- 7
mand need not be supported on block-mode terminals. 7

5.10.7.2.37 write

Synopsis : [*range*] `w[rite][!]` [>>] [*file*]
Synopsis : [*range*] `w[rite]` [!] [*file*]
Synopsis : [*range*] `wq[!]` [>>] [*file*] 7

Write the specified lines (the whole buffer, if no *range* is given) out to the file represented by the pathname *file* , writing to standard output the number of lines and bytes written.

If *file* is specified and is not the current file, and the file named by *file* exists, then 7
the write shall fail. If the current file has been changed by the `file` command, and that file exists, the write shall fail. In either case, the write can be forced by appending the character ! to the command name. An existing file can be appended to by appending >> to the command name. If the file does not exist, the result is implementation defined.

If the *file* is preceded by !, the program named in the **SHELL** environment variable shall be invoked with *file* as its second argument, and the specified lines shall be passed as standard input to the command. The ! in this usage shall be separated from the `w` command by at least one `<blank>` character. The special meaning of the ! can be overridden by escaping it with a backslash character. This command shall be affected by the editor options `writeany` and `readonly`.

The command `wq` shall be equivalent to a `w` followed by a `q`; `wq!` shall be equivalent to `w!` followed by `q`. If the current buffer has no pathname associated with it, the `write` command shall fail.

5.10.7.2.38 xit

Synopsis : [*range*] `x[it][!]` [*file*]

Perform a `write` command (see 5.10.7.2.37) if any changes have been made to the current buffer since the last write to any file.

Unless the command fails because an attempt to write lines to a file did not succeed, the `ex` utility shall exit after an `x` command. This command shall be affected by the editor options `writeany` and `readonly`.

5.10.7.2.39 yank

Synopsis : [*range*] `ya[nk]` [*buffer*] [*count*]

Place the specified lines in the named buffer. If no buffer is specified, the unnamed buffer shall be used (where the most recently deleted or yanked text is placed by default).

5.10.7.2.40 Adjust Window

Synopsis : [*line*] `z` [*type*] [*count*] [*flags*]

If *type* is omitted, then *count* lines following the specified line (*line*) shall be written. The default for *count* shall be the value of the editor option `window`. The *type* argument shall change the position at which *line* shall be written on the screen by affecting the number of lines written before and after *line* .

If *type* is specified, it shall be one of the following:

– Place *line* at the bottom of the screen.

+ Place *line* at the top of the screen.

- `.` Place *line* in the middle.
- `^` Write out *count* lines starting *count* *2 lines before the addressed line; the net effect of this shall be that a `z^` command following another `z` command writes the previous page.
- `=` Center the addressed line on the screen with a line of hyphens written immediately before and after it. The number of preceding and following lines of text written shall be reduced to account for these lines of hyphens.

In all cases the current line indicator shall be set to the last line written, with the exception of the `=` type, which shall cause the current line indicator to be set to that addressed in the command.

5.10.7.2.41 Escape

Synopsis : `!` *command*

Synopsis : [*range*]`!` *command*

Pass the remainder of the line after the `!` character to the program named in the **SHELL** environment variable for execution. A warning shall be issued if the buffer has been changed since the last write. A single `!` character shall be written when the command completes. The current line position shall not be affected.

Within the text of *command* , `%` and `#` shall be expanded as pathnames (the current and alternate pathnames, respectively), and `!` shall be replaced with the text of the previous `!` command. (Thus, `!!` shall repeat the previous `!` command.) If any such expansion is done, the expanded line shall be echoed.

The special meanings of `%`, `#`, and `!` can be overridden by escaping them with a backslash character. This command shall be affected by the editor options `autowrite` and `writeany`; see 5.10.7.5 for details.

In the second form of the `!` command, the remainder of the line after the `!` shall be passed to the program named in the **SHELL** environment variable, as described above. The specified lines shall be provided to the program as standard input; the resulting output shall replace the specified lines.

5.10.7.2.42 Shift Left

Synopsis : [*range*] `<` [*count*] [*flags*]

Shift the specified lines to the left; the number of character positions to be shifted shall be determined by the editor option `shiftwidth`. Only leading <blank>s shall be lost in shifting; other characters shall not be affected. The current line indicator shall be set to the last line changed.

5.10.7.2.43 Shift Right

Synopsis : [*range*] `>` [*count*] [*flags*]

Shift the specified lines to the right, by inserting <blank>s, using <tab>s where possible, as determined by the editor option `shiftwidth`. Empty lines shall not 7 be changed. The current line indicator shall be set to the last line changed. 7

5.10.7.2.44 Resubstitute

Synopsis : [*range*] & [*options*] [*count*] [*flags*]
Synopsis : [*range*] s[ubstitute] [*options*] [*count*] [*flags*]
Synopsis : [*range*] ~ [*options*] [*count*] [*flags*] 8

Repeat the previous substitute command, as if & were replaced by the previous s/*pattern* /*repl* / command. (The same effect can obtained by omitting the /*pattern* /*repl* / string in the substitute command.) The version of the com- 8
mand using tilde shall be the same as & and s, but the *pattern* used shall be the 8
last regular expression used in any command, not necessarily the one used in the 8
last substitute command. For example, in the sequence 8

```
s/red/blue/
/green
~
```

the ~ is equivalent to 8

```
s/green/blue/
```

5.10.7.2.45 Scroll

Synopsis : *eof*

Write the next *n* lines, where *n* is the value of the editor option scroll. The command is invoked with the *eof* character (see the description of the stty eof character in 4.59). The current line indicator shall be set to the last line written. This command need not be supported on block-mode terminals.

5.10.7.2.46 Write Line Number

Synopsis : [*line*] = [*flags*]

Write the line number of the specified line (default last line). The current line position shall not be affected.

5.10.7.2.47 Execute

Synopsis : @ *buffer*
Synopsis : * *buffer*

Execute each line of the named buffer as an ex command. If no buffer is specified or is specified as @ or *, the last buffer executed shall be used. If there is no last buffer, an error occurs.

5.10.7.3 Regular Expressions

The ex utility shall support the basic regular expressions described in 2.8.3. A null RE (//) shall be equivalent to the last RE encountered.

Regular expressions can be used in addresses to specify lines and, in some commands (for example, the `substitute` command), to specify portions of a line to be substituted.

The following constructs can be used to enhance the basic regular expressions:

\ < Match the beginning of a *word*. (See the definition of *word* at the beginning of 5.10.7.2.)

\ > Match the end of a *word*.

~ Match the replacement part of the last `substitute` command. The tilde (~) character can be escaped in a regular expression to become a normal character with no special meaning.

When the editor option `nomagic` is set, the only characters with special meanings shall be `^` at the beginning of a pattern, `$` at the end of a pattern, and `\`. The characters `.`, `*`, `[`, and ~ shall be treated as ordinary characters unless preceded by a \; when preceded by a \ they shall regain their special meaning.

5.10.7.4 Replacement Strings

The character & (\& if the editor option `nomagic` is set) in the replacement string shall stand for the text matched by the pattern to be replaced. The character ~ (\~ if `nomagic` is set) shall be replaced by the replacement part of the previous substitute command. The sequence *n*, where *n* is an integer, shall be replaced by the text matched by the pattern enclosed in the *n*th set of parentheses `\(` and `\)`.

The strings `\l`, `\u`, `\L`, and `\U` can be used to modify the case of elements in the replacement string (using the `\&` or `\`<*digit*>) notation. The string `\l` (`\u`) shall cause the first character (actually inserted by the substitution) that follows the `\l` (`\u`) to be converted to lowercase (uppercase). The strings `\L` (`\U`) shall cause all characters subsequent to them to be converted to lowercase (uppercase) as they are inserted by the substitution until the string `\e` or `\E`, or the end of the replacement string, is encountered.

In visual mode, a `control-Vcontrol-M` (or `control-Qcontrol-M`) sequence in 8
the replacement string shall be mapped to a `<newline>`, and so can be used to 7
split lines. A literal `control-M` requires escaping by preceding it with a 7
backslash (`\^V^M` or `\^Q^M`). 8

5.10.7.5 Edit Options

The `ex` utility has a number of options that modify its behavior. These options have default settings, which can be changed using the `set` command.

Options are Boolean unless otherwise specified.

5.10.7.5.1 **`autoindent, ai`**

[Default *off*]

If `autoindent` is set, each line in input mode shall be indented (using first as many `<tab>`s as possible, as determined by the editor option `tabstop`, and then using `<space>`s) to align with the previous line. (Starting indentation shall be determined by the line appended after, or the line inserted before, or the first line changed, with an `a`, `i`, or `c` command, respectively.) When a `<newline>` character is inserted in the middle of a line, and when `autoindent` is on, the first non-`<blank>` character to the right of the cursor shall be aligned to the current margin on a new line immediately following the current line. Any `<blank>`s to the left of the cursor position at which the `<newline>` was entered shall be retained. When `autoindent` is off, a new line shall be created, but no `<blanks>` shall be discarded. Additional indentation can be provided as usual; succeeding lines shall automatically be indented to the new alignment. A line entered during input mode with `autoindent` that contains no user-entered characters shall be empty, despite the appearance of indentation during entry.

Reducing the indent can be achieved when the cursor is at the current left margin by typing `control-D` one or more times; the cursor shall be moved back to the previous integral number of `shiftwidth` spaces for each `control-D`. A ˆ followed by a `control-D` shall remove all indentation temporarily (for the current line); a `0` followed by a `control-D` shall remove all indentation permanently (for the current line and subsequent lines until input mode is reentered or until the indentation is specifically set to some other value). Changing the indent with `control-D` need not be supported on block-mode terminals.

5.10.7.5.2 **`autoprint, ap`**

[Default *on*]

If `autoprint` is set, the current line shall be written after each command that changes buffer text. (Autoprint shall be suppressed in the `global` [`g` and `v`] commands and for any command on which print operands [*flags*] are used to write explicitly the current line.)

5.10.7.5.3 **`autowrite, aw`**

[Default *off*]

If `autowrite` is set, when a `next`, `rewind`, `tag`, `edit`, `suspend`, `stop`, or `!` command is given, the buffer shall be written (to the current file) if it has been modified. Appending the character `!` to the command name for any of these commands except `!` shall cause the write not to occur. If the write fails, the command shall be aborted and an error message shall be written.

5.10.7.5.4 **`beautify, bf`**

[Default *off*]

If `beautify` is set, all nonprintable characters, other than `<tab>`, `<newline>`, 7
and `<form-feed>`, shall be discarded from text read in from files. 7

5.10.7.5.5 **`errorbells, eb`**

[Default *off*]

If `errorbells` is set, error messages shall be preceded by an alert action. Setting this option off shall cause the user to be informed of an error even when in visual mode, but rather than using the `<alert>` character, an error message shall be written, using a standout mode of the terminal (such as inverse video) instead of the the normal effect of the `<alert>` character, when the effect of the `<alert>` character is to cause the terminal to ring a bell or make other sounds. The editor should place the error message in a standout mode of the terminal, such as inverse video instead of ringing the bell, when the terminal capabilities allow this.

5.10.7.5.6 **`exrc`**

[Default *off*]

If `exrc` is set, `ex` shall access any `.exrc` file in the current directory, as described previously. If `exrc` is not set, `ex` shall ignore any `.exrc` file in the current directory during initialization, unless the current directory is that named by the **HOME** variable.

5.10.7.5.7 **`ignorecase, ic`**

[Default *off*]

If `ignorecase` is set, characters that have uppercase and lowercase representations shall have those representations considered as equivalent for purposes of regular expression comparison.

5.10.7.5.8 **`list`**

[Default *off*]

If `list` is set, write the addressed lines in a way that should be unambiguous:
nonprintable characters shall be written as implementation-defined multicharac- 7
ter sequences; the end of the line shall be marked with a `$`.

5.10.7.5.9 **`magic`**

[Default *on*]

If `magic` is set, change the interpretation of characters in regular expressions and substitution replacement strings (see 5.10.7.3 and 5.10.7.4).

5.10.7.5.10 **mesg**

[Default *on*]

If `mesg` is set, the permission for others to use the `write` or `talk` commands to 7
write to the terminal shall be turned on while in visual mode. The shell-level 7
command `mesg n` (see 5.17) shall take precedence over any setting of the `ex mesg` 7
option; i.e., if `mesg y` was issued before `ex` started (or in a shell escape, such as 7
``:!mesg y''), the `mesg` option in `ex` can suppress incoming messages, but the 7
`mesg` option cannot enable incoming messages if `mesg n` was issued. 7

5.10.7.5.11 **number, nu**

[Default *off*]

If `number` is set, lines shall be written with line numbers, as does the `number` command.

5.10.7.5.12 **paragraphs, para**

[Default *implementation defined*]

The `paragraph` option shall define additional paragraph boundaries for the `{` and `}` commands in visual mode. The `paragraph` option can be set to a character string consisting of zero or more character pairs. The default value is implementation defined.

In the text to be edited, the character string `<newline>.`*<char-pair>*, where *<char-pair>* is one of the character pairs found in `paragraph`, shall define a paragraph boundary. For example, if

```
paragraph=LaA ##
```

then all of the following additional paragraph boundaries would be recognized:

```
<newline>.La
<newline>.A<space>
<newline>.##
```

5.10.7.5.13 **prompt**

[Default *on*]

If `prompt` is set, command mode input shall be prompted for with a colon (`:`); when unset, no prompt shall be written.

5.10.7.5.14 **readonly**

[Default *see text*]

If `readonly` is set, read-only mode shall be enabled. Writing to a different file shall be allowed in read-only mode; in addition, the write can be forced by using the character ! (see the editor command `write`). The default setting shall be *off* unless the file lacks write permission or the command-line option `−R` is used.

5.10.7.5.15 **`remap`**

[Default *on*]

If `remap` is set, macro translation shall allow for macros defined in terms of other macros; translation shall continue until the final product is obtained. If unset, only a one-step translation shall be done.

5.10.7.5.16 **`report`**

[Default 5]

The value of this option shall give the number of lines that can be changed by an editor command before a report is generated on the number of lines affected.

5.10.7.5.17 **`scroll`**

[Default `window`/2]

The value of this option shall determine the number of lines scrolled on a *eof* command (see 5.10.7.2.45 and the description of the `stty eof` character in 4.59). Changing the value of `window` in **EXINIT**, one of the `.exrc` files, or with a `set` command shall not affect the value of `scroll`. This editor option need not be supported on block-mode terminals.

5.10.7.5.18 **`sections`**

[Default *implementation defined*]

The `sections` option shall define additional section boundaries for the `[[` and `]]` commands in visual mode. The `sections` option can be set to a character string consisting of zero or more character pairs. The default value is implementation defined.

In the text to be edited, the character string `<newline>.`*<char-pair>*, where *<char-pair>* is one of the character pairs found in `sections`, shall define a section boundary in the same manner that paragraph boundaries are defined. (See 5.10.7.5.12.)

5.10.7.5.19 **`shell, sh`**

[Default from the environment **SHELL**]

The value of this option can be a string representing the pathname of the shell to be invoked for the ! shell escape command, and by the `shell` command. The default shall be taken from the **SHELL** variable in the environment; see 5.10.5.3 for default values for **SHELL**.

5.10.7.5.20 **`shiftwidth, sw`**

[Default 8]

The value of this option shall give the width of an indentation level used during `autoindent` and by the shift commands.

5.10.7.5.21 **`showmatch, sm`**

[Default *off*]

If `showmatch` is set, in visual mode, when a `)` or `}` is typed, the matching `(` or `{` shall be shown if it is still on the screen. This editor option need not be supported on block-mode terminals.

5.10.7.5.22 **`showmode`**

[Default *off*]

If `showmode` is set, in visual mode, the current mode that the editor is in shall be written on the last line of the screen. Modes that shall be reported are command mode and input mode; other unspecified modes may be written.

5.10.7.5.23 **`tabstop, ts`**

[Default 8]

The value of this option shall specify the software tab stops to be used by the editor to expand tabs in the input.

5.10.7.5.24 **`tags`**

[Default *see text*] 7

The value of this option can be a string representing `<space>`-separated pathnames that shall be used as tag files for the `tag` command. A requested tag shall be searched for sequentially in the specified files. By default, filenames of `tags` shall be searched for in the current directory and in other implementation-defined directories.

5.10.7.5.25 **`term`**

[Default from the environment **TERM**]

The value of this option can be a string representing the terminal type of the output device. The default shall be taken from the **TERM** variable in the environment; see 5.10.5.3 for default values for **TERM**.

5.10.7.5.26 **`terse`**

[Default *off*]

If `terse` is set, error messages may be less verbose. However, except for this caveat, error messages are unspecified. Furthermore, not all error messages need change for different settings of this option.

5.10.7.5.27 `warn`

[Default *on*]

If `warn` is set, `ex` shall write a warning message to standard error if the contents of the buffer have not been saved before a ! command escape.

5.10.7.5.28 `window`

[Default *see text*]

The value of this option shall determine the default number of lines in a screenful, as written by the `z` command. When in visual mode, the number of lines output when moving up or down the file by a *screenful* (see 5.35). The value of `window` can be unrelated to the real screen size, except that it shall be set on entry to
be the current number of screen lines. (The current number of screen lines shall 7
be determined by the system or overridden by the user, as described for **LINES** in 7
5.10.5.3 and 2.6.) The baud rate of the terminal line may reduce the default in an 7
implementation-defined manner. The default value of `windows` also can be over- 7
ridden by specifying a window size using the `−w` command-line option. 7

5.10.7.5.29 `wrapscan, ws`

[Default *on*]

If `wrapscan` is set, searches (using `//` or `??`) shall wrap around the end of the editing buffer; when unset, searches shall stop at the beginning of the editing buffer for `??`, or at the end of the editing buffer for `//`.

5.10.7.5.30 `wrapmargin, wm`

[Default 0]

If the value of this option is greater than zero (say *n*) in visual mode during text
entry, then, in the POSIX Locale, a `<newline>` shall replace all consecutive 7
`<blank>`s, at the boundary between a `<blank>` and a non-`<blank>`, so that lines 7
shall end at least *n* spaces from the ending margin of the terminal screen. (The 7
ending margin shall be determined by the system or overridden by the user, as 7
described for **COLUMNS** in 5.10.5.3 and 2.6.) If a line consists of a sequence of 7
`<non->`blank characters long enough such that it extends continuously from the beginning margin to beyond the ending margin, that sequence shall not be broken by the action of this option. If the value is zero, no wrapping shall be performed.

5.10.7.5.31 `writeany, wa`

[Default *off*]

If `writeany` is set, file-overwriting checks shall be inhibited that would otherwise 7
be made before `write` and `xit` commands, or before an automatic write (see editor option `autowrite`), allowing a write to any file (provided permissions allow it).

5.10.8 Exit Status

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

0 Successful completion.

>0 An error occurred.

5.10.9 Consequences of Errors

When an error in the input script is encountered, or when an error is detected 7
that is a consequence of the data (not) present in the file or due to an external 7
condition such as a read or write error: 7

— If the standard input is a terminal device file, all input shall be flushed, 8
and a new command read. 8

— If the standard input is a regular file, `ex` shall terminate with a nonzero 7
exit status. 7

— If the command in error was one of those specified with the command-line 7
`−c` option (or the obsolescent `+` option), all of the remaining `−c` commands 7
shall be flushed, and a new command read from standard input. 7

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

Usage, Examples

All of the following are valid *addresses* :

`+++` three lines after the current line

`/`*re*`/-` one line before the next occurrence of *re*

`-2` two lines before the current line

The rules for case-insensitive matching (editor option `ignorecase`) may lead to counter-intuitive situations when uppercase characters are used in range expressions. Historical versions of this text implied that the process was:

(1) Take a line of text from buffer

(2) Convert uppercase to lowercase in text line

(3) Convert uppercase to lowercase in RE, except in character class 7
specifications 7

(4) Match RE against text

This would mean that, with `ignorecase` in effect, the text

```
The cat sat on the mat
```

would be matched by

```
/^the/
```

but not by

```
/^[A-Z]he/
```

If a SIGSEGV signal is received while `ex` is saving a file, the file might not be successfully saved.

The `next` command can accept more than one file, so usage such as

```
next 'ls [abc]*'
```

is valid; it would not be valid for the `edit` or `read` commands, for example, because they expect only one file and unspecified results occur.

An example of case conversion with the `s` command:

```
:p
The cat sat on the mat.
:s/\<.at\>/\u&/gp
The Cat Sat on the Mat.
:s/S\(.*\)M/S\U\1\eM/p
The Cat SAT ON THE Mat.
```

The following table is a condensed version of information contained in the normative text. It is presented here to facilitate the review for accuracy of the `ex` commands and the editor options that affect those commands.

`ex` Command	Editor Option	Description
`a[ppend]`	`autoindent`	Determines starting indentation by the line immediately preceding the append.
`a[ppend]!`	`autoindent`	`a!` causes the autoindent editor option setting to be toggled for the duration of this command only.
`c[hange]`	`autoindent`	Determines starting indentation based on the first line that was changed.
`c[hange]!`	`autoindent`	`c!` causes the autoindent editor option setting to be toggled for the duration of this command only.
`e[dit]` `ex`	`autowrite`	Causes the buffer to be written to the current file if it has been modified. (`e!` or `ex!` causes write not to occur.)
	`writeany`	Inhibits checks before an automatic write, allowing a write to any file (provided that permissions allow it).
`g[lobal]` `v`	`autoindent`	Editor option is inhibited when using this command.
	`autoprint`	Editor option is inhibited when using this command.

	`ignorecase`	Characters that have uppercase and lowercase representations shall have those representations considered as equivalent for purposes of regular expression comparison.
	`magic`	When set, the characters `^ $ . * [ ~` have special meaning in regular expressions. When unset, only characters `^ $` have special meaning in regular expressions.
	`report`	Editor option is inhibited when using this command.
`i[nsert]`	`autoindent`	Determines starting indentation based on the line that is inserted before.
`i[nsert]!`	`autoindent`	`i!` causes the autoindent editor option setting to be toggled for the duration of this command only.
`n[ext]`	`autowrite`	Causes the buffer to be written to the current file if it has been modified. (`n!` causes write not to occur.)
	`writeany`	Inhibits checks before an automatic write, allowing a write to any file (provided that permissions allow it).
`p[rint]`	`list`	Causes nonprintable characters to be written, and marks the end of line with $. 7 7
	`number`	Causes lines to be written with a line number before the text.
`rew[ind]`	`autowrite`	Causes the buffer to be written to the current file if it has been modified. (`rew!` causes write not to occur.)
	`writeany`	Inhibits checks before an automatic write, allowing a write to any file (provided that permissions allow it).
`sh[ell]`	`shell`	The pathname of the shell to be invoked.
`s[ubstitute]`	`ignorecase`	Characters that have uppercase and lowercase representations shall have those representations considered as equivalent for purposes of regular expression comparison.
	`magic`	When set, the characters `^ $ . * [ ~` have special meaning in regular expressions. When unset, only characters `^ $` have special meaning in regular expressions.
	`report`	Determines the number of lines that can be changed by an editor command before a report is generated.
	`wrapscan`	When set, searches (using `//` or `??`) shall wrap

		around the end of the file. When unset, searches shall stop at the beginning of the file for `??`, or at the end of the file for `//`.
`su[spend]` `st[op]`	`autowrite`	Causes the buffer to be written to the current file if it has been modified. (`su!` or `st!` causes write not to occur.)
`ta[g]`	`autowrite`	Causes the buffer to be written to the current file if it has been modified. (`ta!` causes write not to occur.)
	`tags`	File searched when looking for a tag.
	`writeany`	Inhibits checks before an automatic write, allowing a write to any file (provided that permissions allow it).
`vi[sual]`	`window`	Default window size used when invoking `vi` command.
`w[rite]`	`readonly`	Prohibits write to file unless w! is entered.
	`writeany`	Inhibits checks before a write, allowing a write to any file (provided that permissions allow it).
`x[it]`	`readonly`	Prohibits write to file unless `x!` is entered.
	`writeany`	Inhibits checks before an exit, allowing a write to any file (provided that permissions allow it).
`z`	`window`	Default screen size if not explicitly stated.
`!` (Escape)	`autowrite`	Causes the buffer to be written to the current file if it has been modified.
	`shell`	The pathname of the shell to be invoked.
	`warn`	Warn user before a ! command escape if there has been ‘‘[No write since last change]’’
	`writeany`	Inhibits checks before an automatic write, allowing a write to any file (provided that permissions allow it).
`<`	`shiftwidth`	Determines the number of character positions to be shifted.
`>`	`shiftwidth`	Determines the number of character positions to be shifted.
eof	`scroll`	Determines the number of lines to scroll.

The `−s` option (and its obsolescent single-hyphen form) suppresses all interactive user feedback and is useful for editing scripts in batch jobs. The list of specific effects is derived from historical practice. The terminal type ‘‘incapable of supporting visual mode’’ historically has been named `dumb`.

The uses described for `control-V` can also be accomplished with `control-Q`, which is useful on terminals that use `control-V` for the down-arrow function. However, most historical implementations use `control-Q` for the *termios* START character, so the editor will generally not receive the `control-Q` unless `stty −ixon` mode is set. Any of the command characters described in this standard

can be made ineffective by their selection as *termios* control characters, using the 8
`stty` utility or other methods described in POSIX.1 {8}. 8

History of Decisions Made

It is recognized that `ex` visual mode (`vi`) and related features would be difficult, if not impossible, to implement satisfactorily on a synchronous terminal, thus it is not a mandatory requirement that such features should work on all terminals. It is the intention, however, that an `ex` implementation should provide the full set of capabilities on all terminals capable of supporting them.

A conforming syntax has been provided for `ex`.

A ''restricted editor'' (both the historical `red` utility and modifications to `ex`) were considered and rejected for inclusion. Neither option provided the level of security that many users might expect.

The `−c` replacement for ''+command'' was inspired by the `−e` option of `sed`.

The `−l` command for LISP mode was omitted as it was considered that it would be difficult to justify the inclusion of such programming language-dependent features for one language and not for others. Similarly the `lisp` edit mode option was omitted.

The `−t` option was introduced because the `ctags` command appears in POSIX.2 and the option is already widely available.

The `−x` option for encryption, and the associated `crypt` utility, were omitted because the algorithm used was not specifiable and the export control laws of some nations make it difficult to export cryptographic technology.

It was agreed that the ability to edit files of arbitrary type can be occasionally useful, but it was not felt necessary to mandate that an `ex` implementation be required to handle other than text files.

The paragraph in Input Files, ''By default, . . . ,'' is intended to close a long-standing security hole in `ex` and `vi`, that of the little-known ''modelines.'' This ''feature'' is that whereby any line in the first or last five lines of the file containing the strings `"ex:"` or `"vi:"` (and, apparently `"ei:"` or `"vx:"`) is considered to be a line containing editor commands and `ex` interprets all the text up to the next `:` or `<newline>` as a command. Consider the consequences, for example, of an unsuspecting user using `ex` or `vi` as their editor when replying to a mail message in which a line such as `ex:! rm −rf *:` appeared in the signature lines. It was strongly felt that an editor should not *by default* interpret any lines of a file. If this feature is desired it should be under the user's control either by the use of a command-line option or by placing some control line in a initialization file. This approach has indeed already been adopted by some vendors (HP-UX has the `modelines` option, for instance, which is turned off by default).

The current documentation for `ex` seems to imply a particular implementation of a scheme for recovering files after some ''disaster.'' Such details were eliminated.

Explicit dependence on the ASCII character set has been avoided where possible, hence the use of an ''implementation-defined multicharacter sequence'' for the display of nonprintable characters in preference to the historical usage of, say, `^I` for `<tab>`. Room is left for an implementation to select some other means of displaying characters unambiguously: combinations of octal or hexadecimal values with leading slashes along with the escape sequences in Table 2-13. (In fact, the historical `^I` is not really unambiguous at all, because it could actually be the characters `^` and `I`; allowing this historical practice is why the word ‘‘should’’ is used.) It is recommended that if an implementation chooses a new method for this writing that it match the formats used by `ed` and `sed` in POSIX.2.

The writing of NUL by the `p` command is not specified as a special case because the group did not want to require `ex` to support NUL characters in files and buffers.

The action of the `join` command in relation to its action with white space is only defined for the POSIX Locale as the correct amount of white space after a period varies; in Japanese none is required, in French only a single space, and so on.

References to ‘‘function keys’’ in the `map` command were omitted because the specification was too simple to be generally useful in a portable manner.

The `version` command was not included because no sufficiently general specification of the version information could be devised that would still be useful to a portable user. This command name should be used by suppliers who wish to provide version information about the `ex` and `vi` editors.

The `directory` editor option was omitted because it was implementation-specific.

The `edcompatible` editor option was omitted because it was not (`ed`-compatible).

The default for editor option `exrc` was reversed from common practice because it was perceived as a security hole by both the POSIX.2 and POSIX.6 working groups. The POSIX.2 group does not believe this to be an undue burden because those people who use `.exrc` file in directories other than `$HOME` can retain the old behavior (by setting this option in their **EXINIT** environment variables or their `$HOME/.exrc` files). Note that `noexrc` disables the use of `./.exrc`; `exrc` enables it. The default is to disable it.

The `paragraph` and `section` options are in the standard, even though their historical defaults are very oriented to the UNIX system `troff` text formatter, because a ‘‘portable user’’ could use the `{`, `}`, `[[`, or `]]` commands in visual mode and have the cursor stop in unexpected places.

The `redraw` and `slowopen` options were omitted because they are implementation dependent.

In historical implementations the default for editor option `window` is arrived at by the following steps:

(1) The initial default is half the number of lines on the terminal screen as defined in a terminal database (typically, *terminfo* or *termcap*).

(2) If dynamic screen size information is available either from environment variables (System V) or the TTY driver (BSD), this information is used

instead.

(3) If the number of lines is erroneous, the default is set to 23.

(4) If the baud rate is less than 2400, the default is set to 16.

(5) If the baud rate is less than 1200, the default is set to 8. This default can then be changed by the editor options below:

`window`	set the window size explicitly and always
`w300`	set `window` if baud rate is less than 1200
`w1200`	set `window` if baud rate is less than 2400
`w9600`	set `window` if baud rate is greater than 2400

The `w300`, `w1200`, and `w9600` options do not appear in this standard because of their lack of documentation and dependence on specific baud rates.

The action taken for signals other than SIGINT , SIGCONT , and SIGHUP is 7
unspecified because some implementations attempt to save the editing buffer in a 7
useful state when other signals are received. 7

5.11 expand — Convert tabs to spaces

5.11.1 Synopsis

`expand [−t` *tablist* `] [`*file* . . . `]`

Obsolescent Version :

`expand [-`*tabstop* `] [-`*tab1* `,`*tab2* `,`. . . `,`*tabn* `] [`*file* . . . `]`

5.11.2 Description

The `expand` utility shall write files or the standard input to the standard output with `<tab>` characters replaced with one or more `<space>` characters needed to pad to the next tab stop. Any `<backspace>` characters shall be copied to the output and cause the column position count for tab stop calculations to be decremented; the column position count shall not be decremented below zero.

5.11.3 Options

The `expand` utility shall conform to the utility argument syntax guidelines described in 2.10.2; the obsolescent version does not.

The following options shall be supported by the implementation:

`−t` *tablist* Specify the tab stops. The argument *tablist* shall consist of a single positive decimal integer or multiple positive decimal integers, separated by `<blank>`s or commas, in ascending order. If a single number is given, tabs shall be set *tablist* column positions apart instead of the default 8. If multiple numbers are given, the tabs shall be set at those specific column positions.

Each tab-stop position *N* shall be an integer value greater than zero, and the list shall be in strictly ascending order. This shall be taken to mean that, from the start of a line of output, tabbing to position *N* shall cause the next character output to be in the (N+1)th column position on that line.

In the event of `expand` having to process a `<tab>` character at a position beyond the last of those specified in a multiple tab-stop list, the `<tab>` shall be replaced by a single `<space>` in the output.

In the obsolescent version, the single number is specified as *tabstop* with a leading minus; multiple tab stops are specified after a leading minus as *tab1* , *tab2* , etc.

5.11.4 Operands

The following operand shall be supported by the implementation:

file The pathname of a text file to be used as input.

5.11.5 External Influences

5.11.5.1 Standard Input

See Input Files.

5.11.5.2 Input Files

Input files shall be text files.

5.11.5.3 Environment Variables

The following environment variables shall affect the execution of `expand`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files), the processing of `<tab>`s and `<space>`s, and for the determination of the width in column positions each character would occupy on a constant-width-font output device.

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.11.5.4 Asynchronous Events

Default.

5.11.6 External Effects

5.11.6.1 Standard Output

The standard output shall be equivalent to the input files with <tab> characters converted into <space>s, as specified by 5.11.2.

5.11.6.2 Standard Error

Used only for diagnostic messages.

5.11.6.3 Output Files

None.

5.11.7 Extended Description

None.

5.11.8 Exit Status

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

0	Successful completion.
>0	An error occurred.

5.11.9 Consequences of Errors

The `expand` utility shall terminate with an error message and nonzero exit status upon encountering difficulties accessing one of the *file* operands.

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

Usage, Examples

The `expand` utility is useful for preprocessing text files (before sorting, looking at specific columns, etc.) that contain <tab>s.

See the definition of *column position* in 2.2.2.31.

The *tablist* option-argument consists of integers in ascending order. Utility Syntax Guideline 8 (see 2.10.2) mandates that `expand` shall accept the integers (within the single argument) separated using either commas or <blank>s.

History of Decisions Made

None.

5.12 `fc` — Process command history list

5.12.1 Synopsis

`fc [−e` *editor* `] [`*first* `[`*last* `]]`

`fc −l [−nr] [`*first* `[`*last* `]]` 8

`fc −s [`*old*=*new* `] [`*first* `]`

5.12.2 Description

The `fc` utility shall list, or shall edit and re-execute, commands previously 8
entered to an interactive `sh` (see 4.56). 8

The command history list shall reference commands by number. The first number 8
in the list is selected arbitrarily. The relationship of a number to its command 8
shall not change except when the user logs in and no other process is accessing 8
the list, at which time the system may reset the numbering to start the oldest 8
retained command at another number (usually 1). When the number reaches an 8
implementation-defined upper limit, which shall be no smaller than the value in 8
HISTSIZE or 32 767 (whichever is greater), the shell may wrap the numbers, 8
starting the next command with a lower number (usually 1). However, despite 8
this optional wrapping of numbers, `fc` shall maintain the time-ordering sequence 8
of the commands. For example, if four commands in sequence are given the 8
numbers 32 766, 32 767, 1 (wrapped), and 2 as they are executed, command 32 767 8
is considered the command previous to 1, even though its number is higher. 8

When commands are edited (when the `−l` option is not specified), the resulting
line(s) shall be entered at the end of the history list and then re-executed by `sh`.
The `fc` command that caused the editing shall not be entered into the history list. 8
If the editor returns a nonzero exit status, this shall suppress the entry into the 8
history list and the command re-execution. Any command line variable assign- 8
ments or redirection operators used with `fc` shall affect both the `fc` command 8
itself as well as the command that results. For example, 8

```
fc -s -1 2>/dev/null
```
8

reinvokes the previous command, suppressing standard error for both `fc` and the 8
previous command. 8

5.12.3 Options

The `fc` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

- `−e` *editor* — Use the editor named by *editor* to edit the commands. The *editor* string is a utility name, subject to search via the **PATH** variable (see 2.6). The value in the **FCEDIT** variable shall be used as a default when `−e` is not specified. If **FCEDIT** is null or unset, `ed` shall be used as the editor.
- `−l` — (The letter ell.) List the commands rather than invoking an editor on them. The commands shall be written in the sequence indicated by the *first* and *last* operands, as affected by `−r`, with each command preceded by the command number.
- `−n` — Suppress command numbers when listing with `−l`.
- `−r` — Reverse the order of the commands listed (with `−l`) or edited (with neither `−l` nor `−s`).
- `−s` — Re-execute the command without invoking an editor.

5.12.4 Operands

The following operands shall be supported by the implementation:

first
last — Select the commands to list or edit. The number of previous commands that can be accessed shall be determined by the value of the **HISTSIZE** variable. The value of *first* or *last* or both shall be one of the following:

- `[+]`*number* — A positive number representing a command number; command numbers can be displayed with the `−l` option.
- −*number* — A negative decimal number representing the command that was executed *number* of commands previously. For example, −1 is the immediately previous command.
- *string* — A string indicating the most recently entered command that begins with that string. If the *old=new* operand is not also specified with `−s`, the string form of the *first* operand cannot contain an embedded equal sign.

When the synopsis form with `−s` is used:

— If *first* is omitted, the previous command shall be used. 7

For the synopsis forms without `−s`: 7

— If *last* is omitted, *last* shall default to *first* .

— If *first* and *last* are both omitted, the previous sixteen commands shall be listed or the previous one command shall be edited (based on the `−l` option).

— If *first* and *last* are both present, all of the commands from 8
first to *last* shall be edited (without `−l`) or listed (with `−l`). 8
Editing multiple commands shall be accomplished by present- 8
ing to the editor all of the commands at one time, each com- 8
mand starting on a new line. If *first* represents a newer com- 8
mand than *last* , the commands shall be listed or edited in 8
reverse sequence, equivalent to using `−r`. For example, the 8
following pairs of commands are equivalent: 8

```
fc -r 10 20          fc    30 40
fc    20 10          fc -r 40 30
```
8
8

— When a range of commands is used, it shall not be an error to 8
specify *first* or *last* values that are not in the history list; `fc` 8
shall substitute the value representing the oldest or newest 8
command in the list, as appropriate. For example, if there are 8
only ten commands in the history list, numbered 1 to 10, 8

```
fc -l
fc 1 99
```
8
8

shall list and edit, respectively, all ten commands. 8

old=new Replace the first occurrence of string *old* in the commands to be 8
re-executed by the string *new* . 8

5.12.5 External Influences

5.12.5.1 Standard Input

None.

5.12.5.2 Input Files

None.

5.12.5.3 Environment Variables

The following environment variables shall affect the execution of `fc`:

FCEDIT This variable, when expanded by the shell, shall be interpreted as the default value for the `-e` *editor* option's *editor* option-argument. If **FCEDIT** is null or unset, `ed` shall be used as the editor.

HISTFILE This variable shall be interpreted as a pathname naming a command history file. If the **HISTFILE** variable is not set, the shell may attempt to access or create a file `.sh_history` in the user's home directory. If the shell cannot obtain both read and write access to, or create, the history file, it it shall use an unspecified mechanism that will allow the history to operate properly. Implementations may choose to disable the history list mechanism for users with appropriate privileges who do not set **HISTFILE** ; the specific circumstances under which this will occur are implementation defined. If more than one instance of the shell is using the same history file, it is unspecified how updates to the history file from those shells interact. As entries are deleted from the history file, they shall be deleted oldest first. It is unspecified when history file entries are physically removed from the history file.

HISTSIZE This variable shall be interpreted as a decimal number representing the limit to the number of previous commands that are accessible. If this variable is unset, an unspecified default ≥ 128 shall be used. The maximum number of commands in the history list is unspecified, but shall be at least 128. An implementation may choose to access this variable only when initializing the history file. Therefore, it is unspecified whether changes made to **HISTSIZE** after the history file has been initialized are effective.

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.12.5.4 Asynchronous Events

Default.

5.12.6 External Effects

5.12.6.1 Standard Output

When the `−l` option is used to list commands, the format of each command in the list shall be as follows:

`"%d\t%s\n"`, *<line number>* , *<command>*

If both the `−l` and `−n` options are specified, the format of each command shall be:

`"\t%s\n"`, *<command>*

If the *<command>* consists of more than one line, the lines after the first shall be displayed as:

`"\t%s\n"`, *<continued-command>*

5.12.6.2 Standard Error

Used only for diagnostic messages.

5.12.6.3 Output Files

None.

5.12.7 Extended Description

None.

5.12.8 Exit Status

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

0 Successful completion of the listing.

>0 An error occurred.

Otherwise, the exit status shall be that of the commands executed by `fc`.

5.12.9 Consequences of Errors

Default.

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

Usage, Examples

Command numbers have no ordering effects; they are like serial numbers. The `–r` 8
option and *–number* operand address the sequence of command execution, regard- 8
less of serial numbers. So, for example, if the command number wrapped back to 8
1 at some arbitrary point, there would be no ambiguity associated with traversing 8
the wrap point. For example, if the command history were: 8

```
32766: echo 1
32767: echo 2
1: echo 3
```
8 8 8

the number `-2` refers to command 32767 because it is the second previous, regard- 8
less of serial number. 8

Since editors sometime use file descriptors as integral parts of their editing, 8
redirecting their file descriptors as part of the `fc` command can produce unex- 8
pected results. For example, if `vi` is the **FCEDIT** editor, the command 8

```
fc -s | more
```
8

will not work correctly on many systems. 8

History of Decisions Made

This utility is based on the KornShell's `fc` built-in.

An earlier draft specified the `–e` option as `[-e editor [old=new]]`, which is not existing practice. Existing practice in `fc` of `[-e editor]` or `[-e - [old=new]]` is acceptable, but not together. To clarify this, a new option `–s` was introduced replacing the `[-e -]`. This resolves the conflict and makes `fc` conform to the Utility Syntax Guidelines.

HISTFILE Users on windowing systems may want to have separate history files for each window by setting **HISTFILE** as follows:

```
HISTFILE=$HOME/.sh_hist$$
```

Some implementations of the KornShell check for the super-user and do not create a history file unless **HISTFILE** is set. This is done primarily to avoid creating unlinked files in the root file system when logging in during single-user mode. **HISTFILE** must be set for the super-user to have history.

HISTSIZE Needed to limit the size of history files. It is the intent of this standard that when two shells share the same history file, commands that are entered in one shell shall be accessible by the other shell. Because of the difficulties of synchronization over a network, the exact nature of the interaction is unspecified.

Consideration was given to omitting the `fc` utility in favor of the command-line 8
editing feature in `sh`. For example, in `vi` editing mode, typing `<ESC>v` is 8
equivalent to 8

```
EDITOR=vi fc
```
8

However, the `fc` utility allows the user the flexibility to edit multiple commands 8
simultaneously (such as `fc 10 20`) and to use editors other than those supported 8
by `sh` for command-line editing. 8

In the KornShell, the alias `r` (‘‘re-do’’) is preset to `fc -e -` (equivalent to the 8
POSIX `fc –s`). This is probably an easier command name to remember than `fc` 8
(‘‘fix command’’), but it does not meet the Utility Syntax Guidelines. Renaming 8
`fc` to `hist` or `redo` was considered, but since this description closely matches his- 8
torical KornShell practice already, such a renaming was seen as gratuitous. 8
Users are free to create aliases whenever odd historical names such as `fc`, `awk`, 8
`cat`, `grep`, or `yacc` are standardized by POSIX. 8

5.13 `fg` — Run jobs in the foreground

This utility shall be provided on systems that support both the User Portability 7
Utilities Option and the POSIX.1 {8} job control option. On other systems it is 7
optional. 7

5.13.1 Synopsis

`fg` [*job_id*]

5.13.2 Description

If job control is enabled (see the description of `set −m` in 3.14.11), the `fg` utility shall move a background job from the current environment (see 3.12) into the foreground.

Using `fg` to place a job into the foreground shall remove its process ID from the 7
list of those ‘‘known in the current shell execution environment’’; see 3.9.3.1.

5.13.3 Options

None.

5.13.4 Operands

The following operand shall be supported by the implementation:

job_id Specify the job to be run as a foreground job. If no *job_id* operand is given, the *job_id* for the job that was most recently suspended, placed in background, or run as a background job shall be used. The format of *job_id* is described in 2.2.2.206.

5.13.5 External Influences

5.13.5.1 Standard Input

None.

5.13.5.2 Input Files

None.

5.13.5.3 Environment Variables

The following environment variables shall affect the execution of `fg`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.13.5.4 Asynchronous Events

Default.

5.13.6 External Effects

5.13.6.1 Standard Output

The `fg` utility shall write the command line of the job to standard output in the following format:

`"%s\n"`, *<command>*

5.13.6.2 Standard Error

Used only for diagnostic messages.

5.13.6.3 Output Files

None.

5.13.7 Extended Description

None.

5.13.8 Exit Status

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

0 Successful completion.

>0 An error occurred.

5.13.9 Consequences of Errors

If job control is disabled, the `fg` utility shall exit with an error and no job shall be placed in the foreground.

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

Usage, Examples

The `fg` utility will not work as expected when it is operating in its own utility execution environment because that environment will have no applicable jobs to manipulate. See the rationale for `bg`. For this reason, `fg` is generally implemented as a shell regular built-in.

History of Decisions Made

The extensions to the shell specified in POSIX.2 and POSIX.2a have mostly been based on features provided by the KornShell. The job control features provided by `bg`, `fg`, and `jobs` are also based on the KornShell. The developers of the standard examined the characteristics of the C-Shell versions of these utilities and found that differences exist. Despite widespread use of the C-Shell, the KornShell versions were selected for this standard to maintain a degree of uniformity with the rest of the KornShell features selected (such as the very popular command-line editing features).

5.14 `file` — Determine file type

5.14.1 Synopsis

`file` *file* ...

5.14.2 Description

The `file` utility shall perform a series of tests on each specified *file* in an attempt to classify it:

(1) If the file is not a regular file, its file type shall be identified. The file types directory, FIFO, block special, and character special shall be identified as such. Other implementation-defined file types may also be identified.

(2) If the file is a regular file, and:

   (a) The file is zero-length, it shall be identified as an empty file.

   (b) The file is not zero-length, `file` shall examine an initial segment of the file and shall make a guess at identifying its contents or whether it is an executable binary file. (The answer is not guaranteed to be correct.)

If *file* does not exist, cannot be read, or its file status could not be determined, the output shall indicate that the file was processed, but that its type could not be determined.

5.14.3 Options

None.

5.14.4 Operands

The following operand shall be supported by the implementation:

file A pathname of a file to be tested.

5.14.5 External Influences

5.14.5.1 Standard Input

None.

5.14.5.2 Input Files

The *file* can be any file type.

5.14.5.3 Environment Variables

The following environment variables shall affect the execution of `file`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.14.5.4 Asynchronous Events

Default.

5.14.6 External Effects

5.14.6.1 Standard Output

In the POSIX Locale, the following format shall be used to identify each operand, *file* specified:

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

The values for <*type*> are unspecified, except that in the POSIX Locale, if *file* is identified as one of the types listed in Table 5-100, <*type*> shall contain (but is not limited to) the corresponding string. Each space shown in the strings shall be exactly one `<space>` character.

If the file named by the *file* operand does not exist, cannot be read, or the status of the file cannot be determined, the string `cannot open` shall be included as part of the <*type*> field, but this shall not be considered an error that affects the exit status.

Table 5-100 – `file` Output Strings

If *file* is a	<*type*> shall contain the string
directory	`directory`
FIFO	`fifo`
block special	`block special`
character special	`character special`
executable binary	`executable`
empty regular file	`empty`
`ar` archive library (see 6.1)	`archive`
extended `cpio` format (see POSIX.1 {8}10.1.2)	`cpio archive`
extended `tar` format (see POSIX.1 {8}10.1.1)	`tar archive`
shell script	`commands text`
C-language source	`c program text`
FORTRAN source	`fortran program text`

5.14.6.2 Standard Error

Used only for diagnostic messages.

5.14.6.3 Output Files

None.

5.14.7 Extended Description

None.

5.14.8 Exit Status

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

- 0 Successful completion.
- >0 An error occurred.

5.14.9 Consequences of Errors

Default.

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

Usage, Examples

Historical systems have used a ‘‘magic file,’’ generally named `/etc/magic`, to help identify file types from their initial segments. Although the general format of the file is well known, it is not portable for more than a few file types; none of the interesting executable-guessing capabilities could be portable without a list of how all systems format their binary files. However, in the interest of some informal interoperability, the format is presented here for informative purposes only.

The file consists of four `<blank>`-separated fields:

offset A number specifying the offset, in bytes, into the file that is to be tested.

type The type of the data to be tested. The possible values for the type include `string`, `short`, `long`, and other data types from the C Standard {7} or other standards. Some types may be followed by a mask specifier of the form &*number*, which is ANDed with the value before any comparisons are done. The mask specifier is octal if preceded by a `0`, hexadecimal if preceded by a `0x`, and decimal otherwise.

value The value to be compared with the value from the file. If the value is a string, it may contain escape characters. If the value is numeric, it may be preceded by a character indicating the operation to be performed. Allowable values include:

- `=` The value from the file must equal the specified value.
- `<` The value from the file must be less than the specified value.
- `>` The value from the file must be greater than the specified value.
- `&` All bits in the specified value must be set in the value from the file.
- `^` At least one of the bits in the specified value must not be set in the value from the file.
- `x` Any value will match.

message The message to be printed if the comparison succeeds. If the string contains a *printf* () format specification, the value from the file is printed, using the message as the format string.

Some file formats contain additional information that is to be printed along with the file type. A line that begins with the character > indicates additional tests and messages to be printed. If the test on the line preceding the first line with a > succeeds, the test specified in all subsequent lines beginning with > are performed, and the messages printed if the tests succeed. The next line that does not begin with a > terminates this action.

Some example entries:

```
0      string   070707    ASCII cpio archive
0      string   !<arch>   portable archive
0      long     0550      executable
>12    long     >0        not stripped
```

History of Decisions Made

The `file` utility can only be required to ‘‘guess’’ at many of the file types because only exhaustive testing can determine some types with certitude. For example, binary data on some systems might match the initial segment of an executable or a `tar` archive. An exhaustive test might require that an executable be executed, or a C program be compiled, before certifying that the guess was correct. This is unreasonable. Since most historical systems generally provide very good guesses, `file` is in POSIX.2 with only the requirement that a best effort be made and that the output contain certain identifiable strings when the guess is made. Even on a very half-hearted system, the mandated output will be no less useful than that from `ls −l`.

The historical `−c` and `−m` options were omitted because the concept of a ‘‘magic file’’ is not portable.

The `−f` option was omitted because the same effect can (and should) be obtained using the `xargs` utility.

Historical versions of the `file` utility attempt to identify the following types of files: symbolic link, directory, character special, block special, socket, `tar` archive, `cpio` archive, SCCS archive, archive library, empty, `compress` output, `pack` output, binary data, C source, FORTRAN source, assembler source, `nroff`/`troff`/`eqn`/`tbl` source, `troff` output, shell script, C-shell script, English text, ASCII text, various executables, APL workspace, compiled terminfo entries, and CURSES screen images. Only those types that are reasonably well specified in POSIX or are directly related to POSIX.2 utilities are listed in the table.

Implementations that support symbolic links are encouraged to use the string `symbolic link` to identify them.

5.15 `jobs` — Display status of jobs in the current session

5.15.1 Synopsis

`jobs [ −l | −p ] [`*job_id* `... ]`

5.15.2 Description

The `jobs` utility shall display the status of jobs that were started in the current shell environment; see 3.12.

When `jobs` reports the termination status of a job, the shell shall remove its pro- 7
cess ID from the list of those ‘‘known in the current shell execution environment’’;
see 3.9.3.1.

5.15.3 Options

The `jobs` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`−l` (The letter ell.) Provide more information about each job listed. This information shall include the job number, current job, process group ID, state, and the command that formed the job.

`−p` Display only the process IDs for the process group leaders of the selected jobs.

By default, the `jobs` utility shall display the status of all stopped jobs, running 7
background jobs, and all jobs whose status has changed and have not been 7
reported by the shell. 7

5.15.4 Operands

The following operand shall be supported by the implementation:

job_id Specifies the jobs for which the status is to be displayed. If no *job_id* is given, the status information for all jobs shall be displayed. The format of *job_id* is described in 2.2.2.206.

5.15.5 External Influences

5.15.5.1 Standard Input

None.

5.15.5.2 Input Files

None.

5.15.5.3 Environment Variables

The following environment variables shall affect the execution of `jobs`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.15.5.4 Asynchronous Events

Default.

5.15.6 External Effects

5.15.6.1 Standard Output

If the `-p` option is specified, the output shall consist of one line for each process ID:

`"%d\n"`, *<process ID>*

Otherwise, if the `-l` option is not specified, the output shall be a series of lines of the form:

`"[%d] %c %s %s\n"`, *<job-number>*, *<current>*, *<state>*, *<command>*

where the fields shall be as follows:

<current> The character `+` identifies the job that would be used as a default for the `fg` or `bg` utilities; this job can also be specified using the *job_id* `"%+"` or `"%%"`. The character `–` identifies the job that

would become the default if the current default job were to exit; this job can also be specified using the *job_id* `"%−"`. For other jobs, this field is a `<space>`. At most one job can be identified with `+` and at most one job can be identified with `-`. If there is any suspended job, then the current job shall be a suspended job. If there are at least two suspended jobs, then the previous job shall also be a suspended job.

<*job-number*>
A number that can be used to identify the process group to the `wait`, `fg`, `bg`, and `kill` utilities. Using these utilities, the job can be identified by prefixing the job number with `%`.

<*state*> One of the following strings (in the POSIX Locale):

`Running` Indicates that the job has not been suspended by a signal and has not exited.

`Done` Indicates that the job completed and returned exit status zero.

`Done(`*code*`)` Indicates that the job completed normally and that it exited with the specified nonzero exit status, *code*, expressed as a decimal number.

`Stopped`
`Stopped (SIGTSTP)`
Either indicates that the job was suspended by the SIGTSTP signal.

`Stopped (SIGSTOP)`
Indicates that the job was suspended by the SIGSTOP signal.

`Stopped (SIGTTIN)`
Indicates that the job was suspended by the SIGTTIN signal.

`Stopped (SIGTTOU)`
Indicates that the job was suspended by the SIGTTOU signal.

The implementation may substitute the string `Suspended` in place of `Stopped`. If the job was terminated by a signal, the format of <*state*> is unspecified, but it shall be visibly distinct from all of the other <*state*> formats shown here and shall indicate the name or description of the signal causing the termination.

<*command*>
The associated command that was given to the shell.

If the `−l` option is specified, a field containing the process group ID shall be inserted before the <*state*> field. Also, more processes in a process group may be output on separate lines, using only the process ID and <*command*> fields.

5.15.6.2 Standard Error

Used only for diagnostic messages.

5.15.6.3 Output Files

None.

5.15.7 Extended Description

None.

5.15.8 Exit Status

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

- 0 Successful completion.
- >0 An error occurred.

5.15.9 Consequences of Errors

Default.

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

Usage, Examples

Both `%%` and `%+` are used to refer to the current job. Both forms are of equal validity—the `%%` mirroring `$$` and `%+` mirroring the output of `jobs`. Both forms reflect historical practice of the KornShell and the C-Shell with job control.

The `−p` option is historical practice in the KornShell and the System V `jsh`. It is the only way to find out the process group of a job because different implementations have different strategies for defining the process group of the job. Usage such as `$(jobs -p)` provides a way of referring to the process group of the job in an implementation-independent way.

The `jobs` utility will not work as expected when it is operating in its own utility execution environment because that environment will have no applicable jobs to manipulate. See the rationale for `bg`. For this reason, `jobs` is generally implemented as a shell regular built-in.

History of Decisions Made

The extensions to the shell specified in POSIX.2 and POSIX.2a have mostly been based on features provided by the KornShell. The job control features provided by `bg`, `fg`, and `jobs` are also based on the KornShell. The developers of the standard examined the characteristics of the C-Shell versions of these utilities and found that differences exist. Despite widespread use of the C-Shell, the KornShell versions were selected for this standard to maintain a degree of uniformity with the rest of the KornShell features selected (such as the very popular command-line editing features).

The output for terminated jobs is left unspecified to accommodate various historical systems. The following formats have been witnessed:

(1) `Killed(`*signal name* `)`

(2) *signal name*

(3) *signal name* `(coredump)`

(4) *signal description* `- core dumped`

Most users should be able to figure these out, although it means that applications will have trouble parsing them.

The calculation of job IDs was not described since this would suggest an implementation, which may impose unnecessary restrictions.

In an earlier draft, a `−n` option was included to ‘‘Display the status of jobs that have changed, exited, or stopped since the last status report.’’ It was removed because the shell always writes any changed status of jobs before each prompt. Thus, `jobs −n` is a no-op.

5.16 `man` — Display system documentation

5.16.1 Synopsis

`man [−k]` *name* ...

5.16.2 Description

The `man` utility shall write information about each of the *name* operands. If *name* is the name of a standard utility, `man` shall at a minimum write a message describing the syntax used by the standard utility, its options, and operands. If more information is available, the `man` utility shall provide it in an implementation-defined manner.

An implementation may provide information for values of *name* other than the standard utilities. Standard utilities that are listed as optional and that are not supported by the implementation either shall cause a brief message indicating that fact to be displayed or shall cause a full display of information as described previously.

5.16.3 Options

The `man` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following option shall be supported by the implementation:

`−k` Interpret *name* operands as keywords to be used in searching a utilities summary database that contains a brief ''purpose'' entry for each standard utility and write lines from the summary database that match any of the keywords. The keyword search shall produce results that are the equivalent of the output of the following command:

```
grep -Ei '
name
name
...
' summary-database
```

(This assumes that the *summary-database* is a text file with a 7
single entry per line; this organization is not required and the 7
example using `grep −Ei` is merely illustrative of the type of 7
search intended.) The ''purpose'' entry to be included in the data- 7
base shall consist of a terse description of the purpose of the 7
utility. 7

5.16.4 Operands

The following operand shall be supported by the implementation:

name A keyword or the name of a standard utility. When `-k` is not 7
specified and *name* does not represent one of the standard utili- 7
ties, the results are unspecified.

5.16.5 External Influences

5.16.5.1 Standard Input

None.

5.16.5.2 Input Files

None.

5.16.5.3 Environment Variables

The following environment variables shall affect the execution of `man`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and in the summary database).

LC_MESSAGES This variable shall determine the language in which messages should be written.

PAGER This variable shall be interpreted as a string representing an output filtering command for writing the output to a terminal. Any string acceptable as a *command_string* operand to the `sh -c` command (see 4.56), shall be valid. When standard output is a terminal device, the manual-
page output shall be piped through the command. If the 7
PAGER variable is null or not set, the command shall be 7
either `more` (see 5.18) or another paginator utility docu- 8
mented in the system documentation. 8

5.16.5.4 Asynchronous Events

Default.

5.16.6 External Effects

5.16.6.1 Standard Output

The `man` utility writes text describing the syntax of the utility *name,* its options, and its operands, or, when `−k` is specified, lines from the summary database. The format of this text is implementation defined.

5.16.6.2 Standard Error

Used only for diagnostic messages.

5.16.6.3 Output Files

None.

5.16.7 Extended Description

None.

5.16.8 Exit Status

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

0	Successful completion.
>0	An error occurred.

5.16.9 Consequences of Errors

Default.

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

Usage, Examples

The historical **MANPATH** variable is not included in POSIX because no attempt is made to specify naming conventions for manual page files, nor even to mandate that they are files at all. In some systems they could be a true database, or a hypertext file, or even fixed strings within the `man` executable. The developers of the standard consider the portability of manual pages to be outside their scope of work (and more appropriate to the working group developing application-installation tools). However, users should be aware that **MANPATH** is implemented on a number of historical systems and that it can be used to tailor the search pattern for manual pages from the various categories (utilities, functions, file formats, etc.) when the system administrator reveals the location and conventions for manual pages on the system.

The keyword search can rely on at least the text of the clause titles from these utility descriptions and the implementation may add more keywords. The term ''clause titles'' refers to the strings such as

```
man - Display system documentation
ps - Report process status
```

History of Decisions Made

It is recognized that, as specified, the `man` utility is only of minimal usefulness. Opinion in the working group was strongly divided as to how much or how little information `man` should be required to provide. It was felt, however, that the provision of some portable way of accessing documentation would aid user portability. The arguments against a fuller specification were:

(1) Large quantities of documentation should not be required on a system cramped for disk space.

(2) The current manual system does not present information in a manner that greatly aids user-portability.

(3) A ''better help system'' is currently an area in which vendors feel that they can add value to their POSIX implementations.

The `−f` option was considered, but due to implementation differences, it was not included in the standard.

The description was changed to be more specific about what has to be displayed for a utility. The working group felt it was insufficient to allow a display of only the Synopsis, without giving a short description of what each option and operand does.

The ''purpose'' entry to be included in the database can be similar to the clause 7
title (less the numeric prefix) from this standard for each utility. These titles are 7
similar to those used in historical systems for this purpose. 7

See 4.40.10 for rationale concerning the default paginator. 7

5.17 `mesg` — Permit or deny messages

5.17.1 Synopsis

```
mesg [ y | n ]
```

5.17.2 Description

The `mesg` utility shall control whether other users are allowed to send messages
via `write`, `talk`, or other utilities to a terminal device. The terminal device 8
affected shall be determined by searching for the first terminal in the sequence of 8
devices associated with standard input, standard output, and standard error, 8
respectively. With no arguments, `mesg` shall report the current state without 8
changing it. Processes with appropriate privileges may be able to send messages
to the terminal independent of the current state.

5.17.3 Options

None.

5.17.4 Operands

The following operands shall be supported by the implementation in the POSIX Locale:

- `y` Grant permission to other users to send messages to the terminal device. 8
- `n` Deny permission to other users to send messages to the terminal device. 8

5.17.5 External Influences

5.17.5.1 Standard Input

None.

5.17.5.2 Input Files

None.

5.17.5.3 Environment Variables

The following environment variables shall affect the execution of `mesg`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages (from the `mesg` utility) are displayed.

5.17.5.4 Asynchronous Events

Default.

5.17.6 External Effects

5.17.6.1 Standard Output

If no operand is specified, `mesg` shall display the current terminal state in an unspecified format.

5.17.6.2 Standard Error

Used only for diagnostic messages.

5.17.6.3 Output Files

None.

5.17.7 Extended Description

None.

5.17.8 Exit Status

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

0 Receiving messages is allowed.

1 Receiving messages is not allowed.

>1 An error occurred.

5.17.9 Consequences of Errors

Default.

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

Usage, Examples

The terminal changed by `mesg` is that associated with the standard input, output, or error, rather than the controlling terminal for the session. This is because users logged in more than once should be able to change any of their login terminals without having to stop the job running in those sessions. This is not a security problem involving the terminals of other users because appropriate privileges would be required to affect another user's terminals.

History of Decisions Made

The method of checking each of the first three file descriptors in sequence until a terminal is found was adopted from System V.

The file `/dev/tty` is not specified for the terminal device because it was thought to be too restrictive. Typical environment changes for the `n` operand are that write permissions are removed for *others* and *group* from the appropriate device. It was decided to leave the actual description of what is done as unspecified because of potential differences between implementations.

The format for standard output is unspecified because of differences between historical implementations. This output is generally not useful to shell scripts (they can use the exit status), so exact parsing of the output is unnecessary.

The mechanism by which the message status of the terminal is changed is unspecified. Therefore, unspecified actions may cause the status of the terminal to change after `mesg` has successfully completed. These actions may include, but are not limited to: another invocation of the `mesg` utility; login procedures; invocation of the `stty` utility; invocation of the `chmod` utility; etc.

5.18 `more` — Display files on a page-by-page basis

5.18.1 Synopsis

`more` [−ceisu] [−n *number*] [−t *tagstring*] [−p *command*] [*file* ...] 8

Obsolescent Version:

`more` [−ceisu] [−n *number*] [+/*command*] [−t *tagstring*] [*file* ...] 8

5.18.2 Description

The `more` utility shall read files and either write them to the terminal on a page-by-page basis or filter them to standard output. If standard output is not a terminal device, all input files shall be copied to standard output in their entirety, 8 without modification. If standard output is a terminal device, the files shall be 8 written a number of lines (one ‘‘screenful’’) at a time under the control of user commands; see 5.18.7.

Certain block-mode terminals do not have all the capabilities necessary to support the complete `more` definition; they are incapable of accepting commands that are not terminated with a <newline>. Implementations that support such terminals shall provide an operating mode to `more` in which all commands can be terminated with a <newline> on those terminals. This mode shall:

— Be documented in the system documentation. 7

— At invocation, inform the user of the terminal deficiency that requires the <newline> usage and provide instructions on how this warning can be suppressed in future invocations.

— Not be required for implementations supporting only fully capable terminals.

— Not affect commands already requiring <newline>s.

— Not affect users on the capable terminals from using `more` as described in this standard.

5.18.3 Options

The `more` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that +/*command* of the obsolescent version uses a nonstandard syntax.

The following options shall be supported by the implementation:

−c If a screen is to be written that has no lines in common with the current screen, or `more` is writing its first screen, `more` shall not scroll the screen, but instead shall redraw each line of the screen in turn, from the top of the screen to the bottom. In addition, if more is writing its first screen, the screen shall be cleared. This option may be ignored on devices that do not have the ability to clear to the end of a line or the end of a screen.

`−e` Exit immediately after writing the last line of the last file in the argument list; see 5.18.7.

`−i` Perform pattern matching in searches without regard to case. See 2.8.2.

`−n` *number* Specify the number of lines per screenful. The *number* argument is a positive decimal integer. The `−n` option shall override any values obtained from the environment.

`−p` *command* 8
`+`*command* (Obsolescent.) 8
For each file examined, initially execute the `more` command in 8
the *command* argument. If the command is a positioning com- 8
mand, such as a line number or a regular expression search, set 8
the current position (see 5.18.7) to represent the final results of 8
the command, without writing any intermediate lines of the file. 8
(For example, the two commands 8

```
more -p 1000j file
more -p 1000G file
```
8 8

would be equivalent and start the display with the current posi- 8
tion at line 1000, bypassing the lines that `j` would write and 8
scroll off the screen if it had been issued during the file examina- 8
tion.) If the positioning command is unsuccessful, the first line in 8
the file shall be the current position. 8

`−s` Replace consecutive empty lines with a single empty line.

`−t` *tagstring*
Write the screenful of the file containing the tag named by the *tagstring* argument. See the `ctags` utility in 5.7. The tags feature represented by `−t` *tagstring* and the `:t` command (see
5.18.7.23) is optional. It shall be provided on any system that 8
also provides a conforming implementation of `ctags`; otherwise, 8
the use of `−t` produces undefined results. 7

`−u` Treat `<backspace>` as a printable control character, displayed as 7
an implementation-defined character sequence (see 5.18.7), 7
suppressing backspacing and the special handling that produces 7
underlined or standout-mode text on some terminal types. Also, 7
do not ignore a `<carriage-return>` character at the end of a 7
line. 7

If both the `−t` *tagstring* and `−c` *command* (or the obsolescent `+/`*command*) 8
options are given, the `−t` *tagstring* shall be processed first; i.e., the file containing 8
the tag is selected by `−t` and then the command is executed. 8

5.18.4 Operands

The following operand shall be supported by the implementation:

file A pathname of an input file. If no *file* operands are specified, the standard input shall be used. If a *file* is −, the standard input shall be read at that point in the sequence.

5.18.5 External Influences

5.18.5.1 Standard Input

The standard input shall be used only if no *file* operands are specified, or if a *file* operand is −.

5.18.5.2 Input Files

The input files being examined shall be text files. If standard output is a termi- 8
nal, standard error shall be used to read commands from the user. If standard 8
output is a terminal, standard error is not readable, and command input is 8
needed, `more` shall terminate with an error indicating that it was unable to read 8
user commands. If standard output is not a terminal, no error shall result if stan- 8
dard error cannot be opened for reading. 8

5.18.5.3 Environment Variables

The following environment variables shall affect the execution of `more`:

COLUMNS This variable shall override the system-selected horizontal 7
screen size. See 2.6 for valid values and results when it is 7
unset or null. 7

EDITOR This variable shall be used by the `v` command to select an editor; see 5.18.7.

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_COLLATE This variable shall determine the locale for character collation information in regular expressions.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files), and the behavior of character classes within regular expressions.

LC_MESSAGES This variable shall determine the language in which messages should be written.

LINES This variable shall override the system-selected vertical 7
screen size, used as the number of lines in a screenful. 7
See 2.6 for valid values and results when it is unset or 7
null. The `-n` option shall take precedence over the **LINES** 7
variable for determining the number of lines in a screen- 7
ful. 7

MORE This variable shall be interpreted as a string containing options described in 5.18.3, preceded with hyphens and
`<blank>`-separated as on the command line. Any 8
command-line options shall be processed after those in the 8
MORE variable, as if the command line were: 8

`more $MORE` *options operands* 8

The **MORE** variable shall take precedence over the **TERM** and **LINES** variables for determining the number of lines in a screenful.

TERM This variable shall be interpreted as the name of the terminal type. If this variable is unset or null, an
unspecified default terminal type shall be used. 7

5.18.5.4 Asynchronous Events

Default.

5.18.6 External Effects

5.18.6.1 Standard Output

The standard output shall be used to write the contents of the input files. 8

5.18.6.2 Standard Error

Used for diagnostic messages and user commands (see 5.18.5.2), and, if standard 8
output is a terminal device, to write a prompting string. The prompting string 8
shall only appear as the last line of the ''screen'' and shall contain the name of the 8
file currently being examined, but it is otherwise unspecified. User input for the 8
`/`, `?`, `:e`, and `:t` commands shall be written on the same line of the screen as the 8
prompt. It is unspecified if informational messages are written for other user 8

commands.

5.18.6.3 Output Files

None.

5.18.7 Extended Description

The number of lines available per ‘‘screen’’ shall be determined by the `-n` option, if present, or by examining values in the environment (see 5.18.5.3). If neither method yields a number, an unspecified number of lines shall be used. The actual number of lines written shall be one less than this number, as the last line of the screen shall be used to write a user prompt and user input. If the number of lines available per screen is less than four, the results are undefined.

In the following descriptions, the ‘‘current position’’ refers to two things:

— The position of the current line on the screen.

— The line number (in the file) of the current line on the screen.

Usually, the line on the screen corresponding to the current position is the third line on the screen. If this is not possible (there are fewer than three lines to display, or this is the first page of the file, or it is the last page of the file), then the current position is either the first or last line on the screen as described later.

The number of columns available per line shall be determined by examining values in the environment (see 5.18.5.3), with a default value as described in 2.6. The `more` utility shall write lines containing more characters than would fit into this number of columns by breaking the line into one or more ‘‘logical’’lines where each of these lines but the last contains the number of characters needed to fill the columns. The logical lines shall be written independently of each other; i.e., commands affecting a single line shall affect them separately.

When standard output is a terminal and `-u` is not specified, `more` shall treat `<backspace>`s and `<carriage-return>`s specially:

— A character, followed first by a `<backspace>`, then by an underscore (_), shall cause that character to be written as underlined text, if the terminal type supports that. An underscore, followed first by a `<backspace>`, then any character, shall also cause that character to be written as underlined text, if the terminal type supports that.

— A `<backspace>` that appears between two identical printable characters shall cause the first of those two characters to be written as emboldened text (i.e., visually brighter, standout mode, or inverse-video mode), if the terminal type supports that, and the second to be discarded. Immediately subsequent occurrences of backspaces/character pairs for that same character shall also be discarded. (For example, the sequence `a\ba\ba\ba` is interpreted as a single emboldened `a`.)

— Other `<backspace>` sequences shall be written directly to the terminal, which generally cause the character preceding the `<backspace>` to be suppressed in the display.

— A `<carriage-return>` at the end of a line shall be ignored, rather than being written as a control character, as described in the next paragraph.

It is implementation defined how other nonprintable characters are written. Implementations should use the same format that they use for the `ex` `print` command; see 5.10.7.2.21.

If the `−t` option was specified, `more` shall write a screen of the file containing the specified tag in the current position.

If the `−p` option was specified, `more` shall find the first occurrence of a line in the input file that contains characters matching the specified pattern and write the screen so that the line found is in the current position.

If neither the `−p` or `−t` options were specified, `more` shall write the first screen of the first input file.

Once the initial screen has been written, `more` shall prompt the user and, based on the interactive user input, shall modify the screen or exit. If the modification of the screen results in a screen that has lines in common with the current screen, `more` shall scroll the screen rather than clearing and redrawing the screen (unless the `−c` option is specified). If the modification of the screen results in a screen that has no lines in common with the current screen, `more` shall clear and redraw the screen.

If the standard output is not a terminal device, `more` shall always exit when it reaches end-of-file on the last file in its argument list. Otherwise, for all files but the last, `more` shall prompt, with an indication that it has reached the end of file, along with the name of the next file. For the last file specified, or for the standard input if no file is specified, `more` shall prompt, indicating end-of-file, and accept additional commands. If the next command specifies forward scrolling, `more` shall exit. If the `−e` option is specified, `more` shall exit immediately after writing the last line of the last file.

Several of the commands described in this clause move the current position backwards in the input stream. In the case that text is being taken from a nonrewindable stream, such as a pipe, it is implementation defined how much backward motion is supported.

If a scrolling command cannot be performed because there are insufficient lines to scroll, `more` shall alert the terminal. The scrolling commands are `b`, `control-B`, `d`, `control-D`, `f`, `control-F`, `g`, `G`, `j`, `k`, `s`, `u`, and `control-U`.

The interactive commands in the following subclauses shall be supported. Some commands can be preceded by a decimal integer, called *count* in the following descriptions. If not specified with the command, *count* shall default to 1.

In the following descriptions, *pattern* is a basic regular expression, as described in 2.8.3. The term ‘‘examine’’ is historical usage meaning ‘‘open the file for viewing’’; for example, `more foo` would be expressed as ‘‘examining’’ file `foo`.

5.18.7.1 Help

Synopsis : `h`

Write a summary of these commands and other implementation-defined commands.

5.18.7.2 Move forward one screenful

7

Synopsis : [*count*]`f`
Synopsis : [*count*]`control-F`

Move forward *count* lines, with a default of one screenful. At end-of-file, `more` 7
shall continue with the next file in the list, or exit if the current file is the last file in the list.

5.18.7.3 Move backward one screenful

Synopsis : [*count*]`b`
Synopsis : [*count*]`control-B`

Move backward *count* lines, with a default of one screenful (see option `−n`). If *count* is more than the screen size, only the final screenful shall be written.

5.18.7.4 Scroll forward one line

Synopsis : [*count*]`<space>` 7
Synopsis : [*count*]`j`
Synopsis : [*count*]`<newline>`

Scroll forward *count* lines. The default *count* for `<space>` shall be one screenful; 7
for `j` and `<newline>`, one line. The entire *count* lines shall be written, even if 7
count is more than the screen size. At end-of-file, `<newline>` shall cause `more` to continue with the next file in the list, or exit if the current file is the last file in the list.

5.18.7.5 Scroll backward one line

Synopsis : [*count*]`k`

Scroll backward *count* lines, with a default of 1. The entire *count* lines shall be written, even if *count* is more than the screen size.

5.18.7.6 Scroll forward one-half screenful

Synopsis : [*count*]`d`
Synopsis : [*count*]`control-D`

Scroll forward *count* lines, with a default of one half of the screen size. If *count* is specified, it shall become the new default for subsequent `d` and `u` commands.

5.18.7.7 Skip forward one line

Synopsis : [*count*]`s`

Skip forward *count* line(s), with a default of 1, and write the next screenful beginning at that point. If *count* would cause the current position to be such that less than one screenful would be written, the last screenful in the file shall be written.

5.18.7.8 Scroll backward one-half screenful

Synopsis : [*count*]`u`
Synopsis : [*count*]`control-U`

Scroll backward *count* lines, with a default of one half of the screen size. If *count* is specified, it shall become the new default for subsequent `d` and `u` commands.

5.18.7.9 Go to beginning of file

Synopsis : [*count*]`g`

Go to line *count* in the file, with a default of 1 (beginning of file). Scroll or rewrite 7
the screen so that that line is at the current position. 7

5.18.7.10 Go to end of file

Synopsis : [*count*]`G`

Go to line *count* in the file, with a default of the end of the file. If no *count* is 8
specified, scroll or rewrite the screen so that the last line in the file is at the bot- 8
tom of the screen. If *count* is specified, scroll or rewrite the screen so that that 8
line is at the current position. 8

5.18.7.11 Refresh the screen

Synopsis : `r`
Synopsis : `control-L`

Refresh the screen.

5.18.7.12 Discard and refresh

Synopsis : `R`

Refresh the screen, discarding any buffered input. If the current file is nonseek- 8
able, buffered input shall not be discarded and the `R` command is equivalent to 8
the `r` command. 8

5.18.7.13 Mark position

Synopsis : m*letter*

Mark the current position with the letter named by *letter*, where *letter* represents
the name of one of the lowercase letters of the portable character set. When a 7
new file is examined, all marks may be lost.

5.18.7.14 Return to mark

Synopsis : '*letter*

Return to the position that was previously marked with the letter named by *letter*, 7
making that line the current position. 7

5.18.7.15 Return to previous position

Synopsis : ''

Return to the position from which the last large movement command was executed (where a ‘‘large movement’’ is defined as any movement of more than a screenful of lines). If no such movements have been made, return to the beginning of the file.

5.18.7.16 Search forward for pattern

Synopsis : [*count*]/[!]*pattern* <newline>

Search forward in the file for the *count*-th line containing the *pattern*. The *count*
shall default to 1. The search shall start at the line following the current position. 7
If the search is successful, the screen shall be modified so that the searched-for 7
line is in the current position. The null regular expression (/<newline>) shall 7
repeat the search using the previous regular expression. If the character ! is
included, the line for searching shall be those that do not contain the *pattern*.

If a match is found for the *pattern*, the logical line containing the pattern shall be
written in the current position. If the matching line is already on the screen, the 7
screen shall be scrolled to make that line the current position; otherwise, a full 7
screen shall be written. If no match is found for the pattern, a message to that 7
effect shall be written as a part of the prompt, and the screen and the current 7
position shall remain unchanged. However, if no match is found and the input is
the standard input, the screen may be scrolled to the last screenful of the input.

5.18.7.17 Search backward for pattern

Synopsis : [*count*]?[!]*pattern* <newline>

Search backward in the file for the *count*-th line containing the *pattern*. The
search shall start at the line immediately before the current position. If the 7
search is successful, the screen shall be modified so that the searched-for line is in 7
the current position. The null regular expression (?<newline>) shall repeat the 7

search using the previous regular expression. If the character ! is included, the line for searching shall be those that do not contain the *pattern*.

If a match is found for the *pattern*, the logical line containing the pattern shall be written in the current position. If the matching line is already on the screen, the screen shall be scrolled to make that line the current position; otherwise, a full screen shall be written. If no match is found for the pattern, a message to that effect shall be written as a part of the prompt, and the screen and the current position shall remain unchanged. However, if no match is found and the input is the standard input, the screen may be scrolled to the last screenful of the input.

5.18.7.18 Repeat search

Synopsis: `[`*count*`]n`

Repeat the previous search for *count*-th line (default 1) containing the last *pattern* (or not containing the last *pattern*, if the previous search was `/!` or `?!`).

5.18.7.19 Repeat search in reverse

Synopsis: `[`*count*`]N`

Repeat the search in the opposite direction of the previous search for the *count*-th line (default 1) containing the last *pattern* (or not containing the last *pattern*, if the previous search was `/!` or `?!`).

5.18.7.20 Examine new file

Synopsis: `:e [`*filename*`]<newline>`

Examine a new file. If the *filename* argument is not specified, the ‘‘current’’ file (see the `:n` and `:p` commands below) from the list of files in the command line shall be re-examined. The *filename* shall be subjected to the process of shell word expansions (see 3.6); if more than a single pathname results, the effects are unspecified. If *filename* is a number sign (#), the previously examined file shall be re-examined. If *filename* refers to a nonseekable file, the results are unspecified.

5.18.7.21 Examine next file

Synopsis: `[`*count*`]:n`

Examine the next file. If a number *count* is specified, the *count*-th next file shall be examined. If *filename* refers to a nonseekable file, the results are unspecified.

5.18.7.22 Examine previous file

Synopsis: `[`*count*`]:p`

Examine the previous file. If a number *count* is specified, the *count*-th previous file shall be examined. If *filename* refers to a nonseekable file, the results are unspecified.

5.18.7.23 Go to tag

Synopsis : `:t` *tagstring* `<newline>`

Go to the supplied *tagstring* and scroll/rewrite the screen with that line in the 7
current position; see the `-t` option. If the `ctags` utility is not supported by the 8
system, the use of `:t` produces undefined results.

5.18.7.24 Invoke editor

Synopsis : `v`

Invoke an editor to edit the current file being examined. If standard input is being examined, the results are unspecified. The name of the editor shall be
taken from the environment variable **EDITOR** , or shall default to `vi`. If **EDITOR** 7
represents either `vi` or `ex`, the editor shall be invoked with options such that the 7
current editor line is the physical line corresponding to the current position in 7
`more` at the time of invocation. For example, either `ex` or `vi` is invoked by speci- 7
fying the editor name and following that with `-c` *linenumber* . It is implementa- 7
tion defined whether line-setting options are passed to editors other than `vi` and 7
`ex`. 7

The file types that can be edited are implementation defined. 7

When the editor exits, `more` shall resume on the current file by rewriting the 7
screen with the current line as the current position. 7

5.18.7.25 Display position

Synopsis : `=`
Synopsis : `control-G`

Write the name of the file currently being examined, the number relative to the total number of files there are to examine, the current line number, the current byte number and the total bytes to write, and what percentage of the file precedes the current position. If `more` is reading from standard input, or the file is shorter than a single screen, some of these items need not be written. All of these items shall reference the first byte of the line after the last line written.

5.18.7.26 Quit

Synopsis : `q`
Synopsis : `:q`
Synopsis : `ZZ`

Exit `more`.

5.18.8 Exit Status

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

0	Successful completion.
>0	An error occurred.

5.18.9 Consequences of Errors

If an error is encountered accessing a file when using the `:n` command, `more` shall attempt to examine the next file in the argument list, but the final exit status shall be affected. If an error is encountered accessing a file via the `:p` command, `more` shall attempt to examine the previous file in the argument list, but the final exit status shall be affected. If an error is encountered accessing a file via the `:e` command, `more` shall remain in the current file and the final exit status shall not be affected.

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

Usage, Examples

The text stating ‘‘from a nonrewindable stream ... implementations may limit the amount of backward motion supported’’ would allow an implementation that permitted no backward motion beyond text already on the screen. It was not possible to require a minimum amount of backward motion that would be effective for all conceivable device types. The implementation should allow the user to back up as far as possible, within device and reasonable memory allocation constraints.

The `−p` allows arbitrary commands to be executed at the start of each file. Examples are:

`more -p G file1 file2`	Examine each file starting with its last screenful.
`more -p 100 file1 file2`	Examine each file starting with line 100 in the current position (usually the third line, so line 98 would be the first line written).
`more -p /100 file1 file2`	Examine each file starting with the first line containing the string `100` in the current position

History of Decisions Made

The `more` utility, available in BSD and BSD-derived systems, was chosen as the prototype for the POSIX.2 file display program since it is more widely available than either the public-domain program `less` or `pg`, a pager provided in System V. The 4.4BSD `more` is the model for the features selected; it is almost fully upward-compatible from the 4.3BSD version in wide use and has become more amenable for `vi` users. Several features originally derived from various file editors, found in both `less` and `pg`, have been added to this specification as they have proved extremely popular with users.

There are inconsistencies between `more` and `vi` that result from historical practice. For example, the single-character commands `h`, `f`, `b`, and `<space>` are screen movers in `more`, but cursor movers in `vi`. These inconsistencies were maintained because the cursor movements are not applicable to `more` and the powerful functionality achieved without the use of the control key justifies the differences in most users' minds.

The tags interface has been included in a program that is not a text editor because it promotes another degree of consistent operation with `vi`. It is conceivable that the paging environment of `more` would be superior for browsing source code files in some circumstances.

The operating mode referred to for block-mode terminals effectively adds a `<newline>` to each synopsis line that currently has none. So, for example, `d<newline>` would page one screenful. The mode could be triggered by a command-line option, environment variable, or some other method. The details are not imposed by POSIX.2 because there are so few systems known to support such terminals. Nevertheless, it was felt that all systems should be able to support `more` given the exception cited for this small community of terminals because, in comparison to `vi`, the cursor movements are few and the command set relatively amenable to the optional `<newline>`s.

Some versions of `more` provide a shell escaping mechanism similar to the `ex !` 7
command. The working group did not feel this was necessary in a paginator, par- 7
ticularly given the wide acceptance of multiple window terminals and job control 7
features. (It chose to retain such features in the editors and `mailx` because the 7
shell interaction also gives an opportunity to modify the editing buffer, which is 7
not applicable to `more`). 7

The `−p` (position) option replaces the + command because of the utility syntax 8
guidelines. In earlier drafts, it took a *pattern* argument, but historical `less` pro- 8
vided the more general facility of a command. It would have been desirable to use 8
the same `−c` as `ex` and `vi`, but the letter was already in use. 8

When the standard output is not a terminal, none of the filter-modification 8
options are effective. This is based on historical practice. For example, a typical 8
implementation of `man` pipes its output through `more −s` to squeeze excess white 8
space for terminal users. When `man` is piped to `lp`, however, it is undesirable for 8
this squeezing to happen. 8

5.19 `newgrp` — Change to a new group

5.19.1 Synopsis

`newgrp [−l] [`*group*`]`

Obsolescent Version :

`newgrp [−] [`*group*`]`

5.19.2 Description

The `newgrp` utility shall create a new shell execution environment with a new real and effective group identification. Of the attributes listed in 3.12, the new shell execution environment shall retain the working directory, file creation mask, and exported variables from the previous environment (i.e., open files, traps, unexported variables, alias definitions, shell functions, and `set` options may be lost). All other aspects of the process environment that are preserved by the *exec* family of functions in POSIX.1 {8} also shall be preserved by `newgrp`; whether other aspects are preserved is unspecified.

A new shell execution environment shall be created whether or not `newgrp` is able to change to the requested new group.

If the POSIX.1 {8} {NGROUPS_MAX} is greater than 1, `newgrp` shall add the specified group to the supplemental groups for the process, if there is room to add another group.

With no operands, `newgrp` shall change the effective group back to the group(s) identified in the user's user entry, and if {NGROUPS_MAX} is greater than 1, shall set the list of supplementary groups to that set in the user's group database entries.

If a password is required for the specified group, and the user is not listed as a member of that group in the group database, the user shall be prompted to enter the correct password for that group. If the user is listed as a member of that group, no password shall be requested. If no password is required for the specified group, it is implementation defined whether users not listed as members of that group can change to that group. Whether or not a password is required, implementation-defined system accounting mechanisms may impose additional authorization restrictions that may cause `newgrp` to exit with an error status.

5.19.3 Options

The `newgrp` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that the obsolescent version uses − in a nonstandard manner.

The following option shall be supported by the implementation:

`-l` (The letter ell.)

`-` (Obsolescent.)
Change the environment to what would be expected if the user actually logged in again.

5.19.4 Operands

The following operand shall be supported by the implementation:

group A group name from the group database or a nonnegative numeric group ID. Either specifies the group ID to which the real and effective group IDs shall be set. If *group* is a nonnegative numeric string and exists in the group database as a group name [see *getgrnam* () in POSIX.1 {8} 9.2.1], the numeric group ID associated with that group name shall be used as the group ID.

5.19.5 External Influences

5.19.5.1 Standard Input

None.

5.19.5.2 Input Files

The file `/dev/tty` shall be used to read a single line of text for password checking, when one is required.

5.19.5.3 Environment Variables

The following environment variables shall affect the execution of `newgrp`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.19.5.4 Asynchronous Events

Default.

5.19.6 External Effects

5.19.6.1 Standard Output

None.

5.19.6.2 Standard Error

Used for diagnostic messages and a prompt string for a password, if one is required.

5.19.6.3 Output Files

None.

5.19.7 Extended Description

None.

5.19.8 Exit Status

If `newgrp` succeeds, the exit status shall be the exit status of the shell. Otherwise, the `newgrp` utility shall exit with the following value:

>0	An error occurred.

5.19.9 Consequences of Errors

The invoking shell may terminate.

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

Usage, Examples

None.

History of Decisions Made

Most historical implementations use one of the *exec* functions to implement the behavior of `newgrp`. Errors detected before the *exec* leave the environment unchanged, while errors detected after the *exec* leave the user in a changed environment. While it would be useful to have `newgrp` issue a diagnostic message to tell the user that the environment changed, it would be inappropriate to require this change to some historical implementations.

The password mechanism is allowed in the group file, but how this would be implemented is not specified.

The newgrp utility was retained in POSIX.2 even given the existence of the multiple group permissions feature in POSIX.1 {8}, because it allows files to be created with a specific group ownership. In some systems, the group ownership is determined by the group of the parent directory in which a file is created, as allowed by
POSIX.1 {8}; on such systems, newgrp does not affect the group ownership in 8
directories where the parent directory dictates it. However, POSIX.1 {8} allows 8
systems to omit this feature; therefore, the newgrp functionality is still required on those systems.

5.20 nice — Invoke a utility with an altered system scheduling priority

5.20.1 Synopsis

nice [−n *increment*] *utility* [*argument* ...]

Obsolescent Version:

nice [−*increment*] *utility* [*argument* ...]

5.20.2 Description

The nice utility shall invoke a utility, requesting that it be run with a different 7
system scheduling priority (see 2.2.2.212). With no options and only if the user 7
has appropriate privileges, the executed utility shall be run with a system 7
scheduling priority that is some implementation-defined quantity less than or 7
equal to the system scheduling priority of the current process. If the user lacks 7
appropriate privileges to affect the system scheduling priority in the requested 7
manner, the nice utility shall not affect the system scheduling priority; in this 8
case, a warning message may be written to standard error, but this shall not 8
prevent the invocation of *utility* or affect the exit status. 8

5.20.3 Options

The nice utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that the obsolescent version allows a multidigit decimal integer as an option name.

The following option shall be supported by the implementation:

`−n` *increment*

`-`*increment* (Obsolescent.)

Specify how the system scheduling priority of the executed utility shall be adjusted. The *increment* option-argument shall be a positive or negative decimal integer that shall be used to modify the system scheduling priority of the executed utility in an implementation-defined manner.

Positive *increment* values shall cause a lower or unchanged system scheduling priority. Negative *increment* values may require appropriate privileges and shall cause a higher or unchanged system scheduling priority.

The system scheduling priority shall be bounded in an implementation-defined manner. If the requested *increment* would raise or lower the system scheduling priority of the executed utility beyond implementation-defined limits, then the limit whose value was exceeded shall be used.

5.20.4 Operands

The following operands shall be supported by the implementation:

utility The name of a utility that is to be invoked. If the *utility* operand names any of the special built-in utilities in 3.14, the results are undefined.

argument Any string to be supplied as an argument when invoking the utility named by the *utility* operand.

5.20.5 External Influences

5.20.5.1 Standard Input

None.

5.20.5.2 Input Files

None.

5.20.5.3 Environment Variables

The following environment variables shall affect the execution of `nice`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

PATH This variable shall determine the search path used to locate the utility to be invoked. See 2.6.

5.20.5.4 Asynchronous Events

Default.

5.20.6 External Effects

5.20.6.1 Standard Output

None.

5.20.6.2 Standard Error

Used only for diagnostic messages.

5.20.6.3 Output Files

None.

5.20.7 Extended Description

None.

5.20.8 Exit Status

If the *utility* utility is invoked, the exit status of `nice` shall be the exit status of 7
utility; otherwise, the `nice` utility shall exit with one of the following values: 7

1–125 An error occurred in the `nice` utility. 7

126 The utility specified by *utility* was found but could not be invoked. 7

127 The utility specified by *utility* could not be found. 7

5.20.9 Consequences of Errors

Default.

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

Usage, Examples

Note that, in the obsolescent version, `-5` is a positive *increment*, while `--5` is a negative *increment*. While this might appear counterintuitive, it is how historical versions of `nice` work.

The only guaranteed portable uses of this utility are:

`nice` *utility* Run *utility* with the default lower system scheduling priority.

`nice -n` *<positive integer> utility*
Run *utility* with a lower system scheduling priority.

On some systems they will have no discernible effect on the invoked utility and on others they will be exactly equivalent.

Historical systems have frequently supported the *<positive integer>* up to 20. Since there is no error penalty associated with guessing a number that is too high, users without access to the system conformance document (to see what limits are actually in place) could use the historical 1 to 20 range or attempt to use very large numbers if the job should be truly low priority.

The system scheduling priority value of a process can be displayed using the command:

```
ps -o nice
```

The `command`, `env`, `nice`, `nohup`, `time`, and `xargs` utilities have been specified to 7
use exit code 127 if an error occurs so that applications can distinguish ‘‘failure to 7
find a utility’’ from ‘‘invoked utility exited with an error indication.’’ The value 7
127 was chosen because it is not commonly used for other meanings; most utilities 7
use small values for ‘‘normal error conditions’’ and the values above 128 can be 7
confused with termination due to receipt of a signal. The value 126 was chosen in 7
a similar manner to indicate that the utility could be found, but not invoked. 7
Some scripts produce meaningful error messages differentiating the 126 and 127 7
cases. The distinction between exit codes 126 and 127 is based on KornShell 8
practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], 8
and uses 126 when any attempt to *exec* the utility fails for any other reason. In 8
System V, the exit status from `nice`, when `nice` detects an error, is 2. In 4.3BSD 7
the status is 1. 7

History of Decisions Made

Due to the text about the limits of the system scheduling priority being implementation defined, `nice` is not required to actually change the priority of the executed command; the limits could be zero differences from the system default, although the implementor is required to document this fact in the conformance document. Because of this, including this utility in POSIX.2 does not require implementations to support something like the traditional *nice*() system call. In particular, a POSIX.2 implementation could run on a nonextended POSIX.1 {8} implementation simply by supplying the following `nice` utility. (The example supports only the nonobsolescent form for illustrative purposes.)

```
#include <stdio.h>
extern int      optind;
extern char     *optarg;
#define USAGE   "Usage: nice [-n incr] <utility> [<arg> ...]\n"

int main(int argc, char *argv[ ]);
{
        int     i;

        while ((i = getopt(argc, argv, "n:")) != EOF) {
                switch (i) {
                    default:
                        fprintf(stderr, USAGE);
                        exit(127);
                    case 'n':  /* ignore increment */
                        break;
                }
        }
        argc -= optind;
        argv += optind;
        if (argc > 0) {
                execvp(argv[0], &argv[0]);
                perror(argv[0]);
        } else {
                fprintf(stderr, USAGE);
        }
        exit(127);
}
```

On systems that choose this approach, `nice` is a benign no-op. On the vast majority of systems that support `nice` historically, it becomes more useful. The developers of the standard considered this a quality of implementation issue and resisted calls to omit `nice` unless it could be guaranteed to have a discernible effect on all systems. The efficacy of `nice` is a legitimate procurement consideration.

The 4.3BSD version of `nice` does not check if *increment* is a valid decimal integer. The command `"nice -x utility"`, for example, would be treated the same as the command `"nice --1 utility"`. If the user does not have appropriate privileges, this will result in a ‘‘permission denied’’ error. This is considered a bug.

When a user without appropriate privileges gives a negative *increment*, System V treats it like the command `"nice -0 utility"`, while 4.3BSD writes a ‘‘permission denied’’ message and does not run the utility. Neither was considered clearly superior, so the behavior was left unspecified.

The C-shell has a built-in version of `nice` that has a different interface from the one described here.

The term ‘‘utility’’ is used, rather than ‘‘command,’’ to highlight the fact that shell compound commands, pipelines, etc., cannot be used. Special built-ins also cannot be used. However, ‘‘utility’’ includes user application programs and shell scripts, not just utilities defined in this standard.

Historical implementations of `nice` provide a priority range of 40 or 41 discrete steps, with the default priority being the midpoint of that range. By default they lower the priority of the executed utility by 10.

Some historical documentation states that the *increment* value must be within a fixed range. This is misleading; the valid *increment* values on any invocation are determined by the current process priority, which is not always the default.

The two limits in a previous draft, {POSIX2_MAX_NICE} and {POSIX2_MIN_NICE}, were removed when the `nice` description was reworded to allow users to exceed the system's range with impunity.

The definition of *system scheduling priority* is not intended to suggest that all 7
processes in a system have priorities that are comparable. Scheduling policy 7
extensions such as the realtime priorities in POSIX.4 make the notion of a single 7
underlying priority for all scheduling policies problematic. Some systems may 7
implement the `nice`-related features to affect all processes on the system, others 7
to affect just the general time-sharing activities implied by POSIX.2, and others 7
may have no effect at all. Because of the use of ‘‘implementation defined’’ in `nice` 7
and `renice`, a wide range of implementation strategies are possible. 7

5.21 nm — Write the name list of an object file

This utility shall be provided on systems that support both the User Portability 7
Utilities Option and the Software Development Utilities Option. On other sys- 7
tems it is optional. 7

5.21.1 Synopsis

`nm [−APv] [ −g | −u ] [−t` *format* `]` *file* . . .

5.21.2 Description

The `nm` utility shall display symbolic information appearing in the object file, executable file, or object-file library named by *file*. If no symbolic information is available for a valid input file, the `nm` utility shall report that fact, but not consider that an error condition.

The default base used when numeric values are written is unspecified.

5.21.3 Options

The `nm` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

`−A` Write the full pathname or library name of an object on each line.

`−g` Write only external (global) symbol information.

`−P` Write information in a portable output format, as specified in 5.21.6.1.

`−t` *format* Write each numeric value in the specified format. The format shall be dependent on the single character used as the *format* option-argument:

- `d` The offset shall be written in decimal.
- `o` The offset shall be written in octal.
- `x` The offset shall be written in hexadecimal.

`−u` Write only undefined symbols.

`−v` Sort output by value instead of alphabetically.

5.21.4 Operands

The following operand shall be supported by the implementation:

file A pathname of an object file, executable file, or object-file library.

5.21.5 External Influences

5.21.5.1 Standard Input

See Input Files.

5.21.5.2 Input Files

The input file shall be an object file, an object-file library whose format is the 7
same as those produced by the `ar` utility for link editing (see 6.1), or an execut- 7
able file. The *nm* utility may accept additional implementation-defined object 7
library formats for the input file. 7

5.21.5.3 Environment Variables

The following environment variables shall affect the execution of `nm`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_COLLATE This variable shall determine the locale for character collation information for the symbol-name and symbol-value collation sequences.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments). 7

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.21.5.4 Asynchronous Events

Default.

5.21.6 External Effects

5.21.6.1 Standard Output

If symbolic information is present in the input file(s), then for each file, or for each 7
member of an archive, the `nm` utility shall write the following information to stan- 7
dard output. By default, the format is unspecified, but the output shall be sorted alphabetically by symbol name.

— Library or object name, if `−A` is specified.

— Symbol name.

— Symbol type, which shall either be one of the following single characters or 7
an implementation-defined type represented by a single character: 7

`A`	Global absolute symbol
`a`	Local absolute symbol
`B`	Global ‘‘bss’’(i.e., uninitialized data space) symbol
`b`	Local bss symbol
`D`	Global data symbol
`d`	Local data symbol
`T`	Global text symbol
`t`	Local text symbol
`U`	Undefined symbol

— Value of the symbol.

— The size associated with the symbol, if applicable.

This information may be supplemented by additional information specific to the implementation.

If the `−P` option is specified, the previous information shall be displayed using the following portable format. The three versions differ depending on whether `-t d`, `-t o`, or `-t x` was specified, respectively:

`"%s: %s %s %d %d\n"`, *<library/object name>*, *<name>*, *<type>*, *<value>*, *<size>*

`"%s: %s %s %o %o\n"`, *<library/object name>*, *<name>*, *<type>*, *<value>*, *<size>*

`"%s: %s %s %x %x\n"`, *<library/object name>*, *<name>*, *<type>*, *<value>*, *<size>*

where *<library/object name>* and its trailing colon and `<blank>`s are omitted when `−A` is not specified and *<size>* and its preceding `<blank>`s are omitted when inapplicable to the symbol type.

5.21.6.2 Standard Error

Used only for diagnostic messages.

5.21.6.3 Output Files

None.

5.21.7 Extended Description

None.

5.21.8 Exit Status

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

0 Successful completion.

>0 An error occurred.

5.21.9 Consequences of Errors

Default.

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

Usage, Examples

Historical implementations of `nm` have used different bases for numeric output and supplied different default types of symbols that were reported. The `−t` *format* option, similar to that used in `od` and `strings`, can be used to specify the numeric base; `−g` and `−u` can be used to restrict the amount of output or the types of symbols included in the output.

History of Decisions Made

The option list was significantly reduced from that provided by historical implementations.

The `nm` description is a subset of both the System V and BSD `nm` utilities with no specified default output. 7

It was recognized that mechanisms for dynamic linking make this utility less meaningful when applied to a library, but the value of `nm` during software development was judged to outweigh other limitations.

The compromise of using `−t` *format* versus using `−d`, `−o`, and other similar options was necessary because of differences in the meaning of `−o` between implementations. The `−o` option from BSD has been provided here as `−A` to avoid collision with the −o from System V (which has been provided here as -t o).

The default output format of nm is not specified because of differences in historical implementations. The P option was added to satisfy balloting objections relating to the use of nm in scripts as well as interactively. Some type of portable format was required. The format given in 5.21.6.1 uses spaces between the fields, which may be any number of <blank>s required to get the columns to line up. The single-character types were selected to match historical practice and the requirement that implementation additions also be single characters make parsing the information easier for shell scripts.

5.22 patch — Apply changes to files

5.22.1 Synopsis

patch [−blNR] [−c | −e | −n] [−d *dir*] [−D *define*] [−i *patchfile*] [−o *outfile*] 8
[−r *rejectfile*] [*file*] 8

5.22.2 Description

The patch utility shall read a source (‘‘patch’’) file containing any of the three forms of difference (‘‘diff’’) listings produced by the diff (see 4.17) utility (normal, context, or in the style of ed; see 4.20) and apply those differences to a file. By default, patch shall read from the standard input.

The patch utility shall attempt to determine the type of the diff listing, unless overruled by a −c, −e, or −n option.

If the patch file contains more than one patch, patch shall attempt to apply each
of them as if they came from separate patch files. (In this case the name of the 7
patch file must be determinable for each diff listing.) 7

5.22.3 Options

The patch utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

−b Save a copy of the original contents of each modified file, before 7
the differences are applied, in a file of the same name with the 7
suffix .orig appended to it. If the file already exists, it shall be 7
overwritten; if multiple patches are applied to the same file, the 7
.orig file shall be written only for the first patch. When the 8
−o *outfile* option is also specified, *file* .orig shall not be created 7
but, if *outfile* already exists, *outfile* .orig shall be created. 7

`−c` Interpret the patch file as a context diff (the output of the utility `diff` when the `−c` or `−C` options are specified).

`−d` *dir* Change the current directory to *dir* before processing as described in 5.22.7.

`−D` *define* Mark changes with the C preprocessor construct:

```
#ifdef define
...
#endif
```

The option-argument *define* shall be used as the differentiating symbol.

`−e` Interpret the patch file as an `ed` script, rather than a `diff` script.

`−i` *patchfile* Read the patch information from the file named by the pathname *patchfile* , rather than the standard input.

`−l` (The letter ell.) Cause any sequence of <blank>s in the diff script to match any sequence of <blank>s in the input file. Other characters shall be matched exactly.

`−n` Interpret the script as a normal diff.

`−N` Ignore patches where the differences have already been applied to 7
the file; by default, already-applied patches shall be rejected. 7

`−o` *outfile* Instead of modifying the files (specified by the *file* operand or the 7
diff listings) directly, write a copy of the file referenced by each 7
patch, with the appropriate differences applied, to *outfile* . Multi- 7
ple patches for a single file shall be applied to the intermediate 7
versions of the file created by any previous patches, and shall 7
result in multiple, concatenated versions of the file being written 7
to *outfile* . 7

8

`−R` Reverse the sense of the patch script; i.e., assume that the diff script was created from the new version to the old version. The `−R` option cannot be used with `ed` scripts. The `patch` utility attempts to reverse each portion of the script before applying it. Rejected differences shall be saved in swapped format. If this option is not specified, and until a portion of the patch file is successfully applied, `patch` attempts to apply each portion in its ''reversed'' sense as well as in its normal sense. If the attempt is successful, the user shall be prompted to determine if the `−R` option should be set.

`−r` *rejectfile* Override the default reject file name. In the default case, the 7
reject file shall have the same name as the output file, with the 7
suffix `.rej` appended to it. See 5.22.7.3. 7

5.22.4 Operands

The following operand shall be supported by the implementation:

file A pathname of a file to patch.

5.22.5 External Influences

5.22.5.1 Standard Input

See Input Files.

5.22.5.2 Input Files

Input files shall be text files.

5.22.5.3 Environment Variables

The following environment variables shall affect the execution of `patch`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).

LC_MESSAGES This variable shall determine the processing of affirmative 7
responses and the language in which messages should be 7
written. 7

LC_TIME This variable shall determine the locale for recognizing the format of file time stamps written by the `diff` utility in a context-diff input file.

5.22.5.4 Asynchronous Events

Default.

5.22.6 External Effects

5.22.6.1 Standard Output

None.

5.22.6.2 Standard Error

Used for diagnostic and informational messages. 7

5.22.6.3 Output Files

The output of the `patch` utility, the save files (`.orig` suffixes) and the reject files (`.rej` suffixes) shall be text files.

5.22.7 Extended Description

A patchfile may contain patching instructions for more than one file; file names 8
shall be determined as specified in 5.22.7.2. When the `−b` option is specified, for each patched file, the original shall be saved in a file of the same name with the suffix `.orig` appended to it.

For each patched file, a reject file may also be created as noted in 5.22.7.3. In the absence of a `−r` option, the name of this file shall be formed by appending the suffix `.rej` to the original file name.

5.22.7.1 Patchfile Format

The patch file shall contain zero or more lines of header information followed by one or more patches. Each patch shall contain zero or more lines of filename identification in the format produced by `diff −c` (see 4.17), and one or more sets of `diff` output, which are customarily called ‘‘hunks.’’

The `patch` utility shall recognize the following expression in the header information:

`Index:` *pathname*	The file to be patched is named *pathname*.

8

If all lines (including headers) within a patch begin with the same leading 7
sequence of `<blank>`s, the `patch` utility shall remove this sequence before 7
proceeding. Within each patch, if the type of diff is context, the `patch` utility shall recognize the following expressions:

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

Each hunk within a patch shall be the `diff` output to change a line range within the original file. The line numbers for successive hunks within a patch shall occur in ascending order.

5.22.7.2 Filename Determination

If no *file* operand is specified, `patch` shall perform the following steps to obtain a pathname:

(1) If the patch contains the strings *** and `---`, the `patch` utility shall test for the existence of each pathname relative to the current directory. If both files exist, the `patch` utility shall assume that no pathname can be obtained from this step.

(2) If the header information contains a line with the string `Index:`, the `patch` utility shall test for the existence of this file relative to the current directory.

(3) If no pathname can be obtained by applying the steps (1) or (2), or if the pathnames obtained do not exist, the `patch` utility shall write a prompt to standard output and request a filename interactively from standard input.

5.22.7.3 Patch Application

If the `−c`, `−e`, or `−n` option is present, the `patch` utility shall interpret information within each hunk as a context diff, an `ed` diff, or a normal diff, respectively. In the absence of any of these options, the `patch` utility shall determine the type of diff based on the format of information within the hunk.

For each hunk, the `patch` utility shall begin to search for the place to apply the patch at the line number at the beginning of the hunk, plus or minus any offset used in applying the previous hunk. If lines matching the hunk context are not found, the `patch` utility shall scan both forwards and backwards at least 1000 bytes for a set of lines that match the hunk context.

If no such place is found, and it is a context diff, then another scan shall take place, ignoring the first and last line of context. If that fails, the first two and last two lines of context shall be ignored and another scan shall be made. Implementations may search more extensively for installation locations.

If no location can be found, the `patch` utility shall append the hunk to the reject file. The rejected hunk shall be written in context-diff format regardless of the format of the patch file. If the input was a normal or `ed`-style diff, the reject file may contain diffs with zero lines of context. The line numbers on the hunks in the reject file may be different than the line numbers in the patch file since they shall reflect the approximate locations for the failed hunks in the new file rather than the old one.

If the type of patch is an `ed` diff, the implementation may accomplish the patching by invoking the `ed` utility.

5.22.8 Exit Status

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

- 0 Successful completion.
- 1 One or more lines were written to a reject file.
- >1 An error occurred.

5.22.9 Consequences of Errors

Patches that cannot be correctly placed in the file shall be written to a reject file.

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

Usage, Examples

The −R option will not work with `ed` scripts because there is too little information to reconstruct the reverse operation.

History of Decisions Made

Some of the functionality in historical `patch` implementations has not been specified. The following documents features present in historical implementations that have not been specified.

The `patch` utility's ability to ''check out'' or ''get'' suitable RCS or SCCS files if the source file is unavailable was deleted because POSIX.2 has not yet, if it ever will, required SCCS or RCS, and, more importantly, because it violates the ''Principle of Least Astonishment'' for users. It is strongly suggested that this functionality, if provided, be under the control of a command-line option.

Another deleted piece of functionality was the ''+'' pseudo-option allowing an additional set of options and patch file operand to be given. This was seen as being insufficiently useful to standardize.

In historical implementations, if the string `Prereq:` appeared in the header, the `patch` utility would search for corresponding version information (the string specified in the header, delimited by `<blank>`s or the beginning or end of a line or the file) anywhere in the original file. This was deleted as too simplistic and insufficiently trustworthy a mechanism to standardize. For example, if

```
Prereq: 1.2
```

were in the header, the presence of a delimited 1.2 anywhere in the file would satisfy the prerequisite.

The following options were dropped from historical implementations of `patch`, as insufficiently useful to standardize.

−b The −b option traditionally provided a method for changing the backup file's name extension from `.orig`. This option has been modified and retained in the current specification.

−F The −F option specified the number of lines of a context diff to ignore when searching for a place to install a patch.

−f The −f option traditionally caused `patch` to not request additional information from the user.

−p The −p option attempted to provide a method of customizing the patch file to local user file hierarchies.

Historical versions of `patch` resolved file names in a more complex fashion than that described by this standard. If no *file* operand was specified, the full pathnames and terminating filenames in any context diff as well as any pathnames specified using the `Index:` keyword in the patch header were checked for existence. If none of these files existed, the user was prompted for the name of a file to patch. Additionally, the −p option could be used to delete some of the pathname components from the context diff and `Index:` keyword pathnames. POSIX simplifies this behavior and does not require the −p option for two reasons. The first is that sites with strong naming and stylistic conventions, or with local and ''official''copies of programs, could easily patch the wrong files with the user being unaware of the error. The second is that the −p option was only useful if the local tree was entirely consistent with the tree from which the patch file was created. This was believed unlikely, particularly in patch files with multiple patches, and the functionality is trivially replaced by creating the patch file at a more reasonable place in the tree.

−r The −r option traditionally provided a method of overriding the default reject file's extension from `.rej`.

−s The −s option traditionally caused `patch` to work silently unless an error occurred.

−x The −x option traditionally set internal debugging flags.

In some file system implementations, the saving of a `.orig` file may produce unwanted results. In the case of 12, 13, or 14-character filenames, on file systems supporting 14-character maximum filenames, the `.orig` file will overwrite the new file. The reject file may also exceed this filename limit. It was suggested, due to some existing practice, that a tilde (~) suffix be used instead of `.orig` and some other character instead of the `.rej` suffix. This was rejected because it is not obvious to the user which file is which. The suffixes `.orig` and `.rej` are clearer and more understandable.

The −b option has the opposite sense in some historical implementations—do not save the `.orig` file. The default case here is to not save the files, making `patch` behave more consistently with the other standard utilities.

The `−w` option in previous drafts was changed to `−l` to match historical practice.

The `−N` option was included because without it, a noninteractive application cannot reject previously applied patches. For example, if a user is piping the output of `diff` into the `patch` utility, and the user only wants to patch a file to a newer version noninteractively, the `−N` option is required.

Changes to the `−l` option description were proposed to allow matching across 8
`<newline>`s in addition to just `<blank>`s. Since this is not historical practice, 8
and since some ambiguities could result, it is suggested that future developments 8
in this area be triggered by another option letter, such as `−L`. 8

5.23 `ps` — Report process status

5.23.1 Synopsis

`ps` [`−aA`][`−G` *grouplist*][`−o` *format*] . . . [`−p` *proclist*][`−t` *termlist*] 7
[`−U` *userlist*]

5.23.2 Description

The `ps` utility shall write information about processes, subject to having the appropriate privileges to obtain information about those processes.

When the `−o` option is not specified, information about processes selected shall be
written in an implementation-defined manner. By default, `ps` shall select all 7
processes with the same effective user ID as the current user and the same con- 7
trolling terminal as the invoker. In all cases, the selection of processes to be 7
reported shall be subject to the constraints of appropriate privileges. 7

5.23.3 Options

The `ps` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation.

`−a` Write information for all processes associated with terminals. 7
Implementations may omit session leaders from this list. 7

`−A` Write information for all processes.

`−G` *grouplist*
Write information for processes whose real group ID numbers are given in *grouplist* . The *grouplist* shall be a single argument in the form of a `<blank>`- or comma-separated list.

−o *format* Write information according to the format specification given in *format*. This is fully described in 5.23.6.1. Multiple −o options can be specified; the format specification shall be interpreted as the <space>-separated concatenation of all the *format* option-arguments.

−p *proclist* Write information for processes whose process ID numbers are given in *proclist*. The *proclist* shall be a single argument in the form of a <blank>- or comma-separated list.

−t *termlist* Write information for processes associated with terminals given in *termlist*. The *termlist* shall be a single argument in the form of a <blank>- or comma-separated list. Terminal identifiers shall be given in an implementation-defined format.

−U *userlist* Write information for processes whose real user ID numbers or login names are given in *userlist*. The *userlist* shall be a single argument in the form of a <blank>- or comma-separated list.

With the exception of −o *format*, all of the options shown are used to select 7
processes. If any are specified, the default list shall be ignored and ps shall select 7
the processes represented by the inclusive OR of all the selection-criteria options. 7

5.23.4 Operands

None.

5.23.5 External Influences

5.23.5.1 Standard Input

None.

5.23.5.2 Input Files

None.

5.23.5.3 Environment Variables

The following environment variables shall affect the execution of ps:

COLUMNS This variable shall override the system-selected horizontal 7
screen size, used to determine the number of text columns 7
to display. See 2.6 for valid values and results when it is 7
unset or null. 7

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

LC_TIME This variable shall determine the format and contents of the date and time strings displayed.

5.23.5.4 Asynchronous Events

Default.

5.23.6 External Effects

5.23.6.1 Standard Output

When the `-o` option is not specified, the standard output format is implementation defined. The `-o` option allows the output format to be specified under user control.

The format specification shall be a list of names presented as a single argument, `<blank>`- or comma-separated. Each variable has a default header. The default header can be overridden by appending an equals sign and the new text of the header. The rest of the characters in the argument shall be used as the header text. The fields specified shall be written in the order specified on the command line, and should be arranged in columns in the output. The field widths shall be selected by the system to be at least as wide as the header text (default or overridden value). If the header text is null, such as `-o user=`, the field width shall be at least as wide as the default header text. If all header text fields are null, no header line shall be written.

The following names shall be recognized in the POSIX Locale: 8

`ruser` The real user ID of the process. This shall be the textual user ID, if it can be obtained and the field width permits, or a decimal representation otherwise.

`user` The effective user ID of the process. This shall be the textual user ID, if it can be obtained and the field width permits, or a decimal representation otherwise.

`rgroup` The real group ID of the process. This shall be the textual group ID, if it can be obtained and the field width permits, or a decimal representation otherwise.

`group` The effective group ID of the process. This shall be the textual group ID, if it can be obtained and the field width permits, or a decimal representation otherwise.

`pid` The decimal value of the process ID.

`ppid` The decimal value of the parent process ID.

`pgid` The decimal value of the process group ID.

`pcpu` The ratio of CPU time used recently to CPU time available in the same period, expressed as a percentage. The meaning of ‘‘recently’’ in this context is unspecified. The CPU time available is determined in an unspecified manner.

`vsz` The size of the process in (virtual) memory in kilobytes as a decimal integer.

`nice` The decimal value of the system scheduling priority of the process. See 5.20.

`etime` In the POSIX Locale, the elapsed time since the process was started, in the form:

[[*dd*`-`]*hh*`:`]*mm*`:`*ss*

where *dd* shall represent the number of days, *hh* the number of hours, *mm* the number of minutes, and *ss* the number of seconds. The *dd* field shall be a decimal integer. The *hh*, *mm*, and *ss* fields shall be two-digit decimal integers padded on the left with zeroes.

`time` In the POSIX Locale, the cumulative CPU time of the process in the form:

[*dd*`-`]*hh*`:`*mm*`:`*ss*

The *dd*, *hh*, *mm*, and *ss* fields shall be as described in the `etime` specifier.

`tty` The name of the controlling terminal of the process (if any) in the same format used by the `who` utility.

`comm` The name of the command being executed (*argv*[0] value) as a string.

`args` The command with all its arguments as a string. The implementation may truncate this value to the field width; it is implementation defined whether any further truncation occurs. It is unspecified whether the string represented is a version of the argument list as it was passed to the command when it started, or is a version of the arguments as they may have been modified by the application. Applications cannot depend on being able to modify their argument list and having that modification be reflected in the output of `ps`.

Any field need not be meaningful in all implementations. In such a case a hyphen (`-`) should be output in place of the field value.

Only `comm` and `args` shall be allowed to contain `<blank>`s; all others shall not. Any implementation-defined variables shall specify in the conformance document if the field may contain `<blank>`s, as well as the default header.

Table 5-101 specifies the default header to be used in the POSIX Locale 7
corresponding to each format specifier. 7

Table 5-101 – `ps` Variable Names and Default Headers

Format Specifier	Default Header	Format Specifier	Default Header
args	COMMAND	ppid	PPID
comm	COMMAND	rgroup	RGROUP
etime	ELAPSED	ruser	RUSER
group	GROUP	time	TIME
nice	NI	tty	TT
pcpu	%CPU	user	USER
pgid	PGID	vsz	VSZ
pid	PID		

5.23.6.2 Standard Error

Used only for diagnostic messages.

5.23.6.3 Output Files

None.

5.23.7 Extended Description

None.

5.23.8 Exit Status

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

0 Successful completion.

>0 An error occurred.

5.23.9 Consequences of Errors

Default.

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

Usage, Examples

The command:

```
ps -o user,pid,ppid=MOM -o args
```

writes the following in the POSIX Locale. The contents of the COMMAND field need not be the same in all implementations, due to possible truncation.

```
  USER    PID    MOM    COMMAND
helene     34     12    ps -o uid,pid,ppid=MOM -o args
```

The `args` format specifier is allowed to produce a truncated version of the command arguments. In some implementations, this information is no longer available when the `ps` utility is executed.

If the field width is too narrow to display a textual ID, the system may use a numeric version. Normally, the system would be expected to choose large enough field widths, but if a large number of fields were selected to write, it might squeeze fields to their minimum sizes to fit on one line. One way to ensure adequate width for the textual IDs is to override the default header for a field to make it larger than most or all user or group names.

There is no special quoting mechanism for header text. The header text is the rest of the argument. If multiple header changes are needed, multiple `-o` options can be used, such as

```
ps -o "user=User Name" -o pid=Process\ ID
```

History of Decisions Made

There is very little commonality between BSD and System V implementations of `ps`. Many options conflict or have subtly different usages. The working group has attempted to select a set of options that seem useful on a wide range of systems and has selected options that either can be implemented on both BSD and System V based systems without breaking the current implementations or where the options are sufficiently similar that any changes would not be unduly painful for users or implementors.

It is recognized that on some systems, especially multilevel secure systems, `ps` may be nearly useless. The default output has therefore been chosen both such that it does not break existing implementations and also that it is likely to provide at least some useful information on most systems.

The major change is the addition of the format specification capability. The motivation for this invention is to provide a mechanism for users to access a wider range of system information, if the system permits it, in a portable manner. The fields chosen to appear in the specification were arrived at after considering what concepts were likely to be both reasonably useful to the ''average'' user and stood a reasonable chance of being implemented on a wide range of systems. Again it is recognized that not all systems will be able to provide all the information and, conversely, some may wish to provide more. It is hoped that the approach adopted will be sufficiently flexible and extensible to accommodate most systems. Implementations may be expected to introduce new format specifiers.

The default output should consist of a short listing containing the process ID, terminal name, cumulative execution time, and command name of each process.

The preference of the group would have been to make the format specification an operand of the `ps` command. Unfortunately, BSD usage precluded this.

At one time a format was included to display the process's environment array. This was deleted because there is no portable way to display it.

The `−o nice` field is named from historical precedent. An implementation may provide an extension to name it something based on ''system scheduling priority.''

The historical `−l` option has been omitted because it could not be specified more precisely. Since the default output is ''written in an implementation-defined manner,'' having a longer version of it required is not useful in a standard. However, mentioning `−l` here in the rationale serves to reserve its meaning and implementors are encouraged to provide it as an extension.

The historical options `−g` and `−u` have had different meanings on various systems. 7
They were changed to their uppercase equivalents to achieve a common definition
in POSIX.2.

The `−A` option is equivalent to the BSD `−g` and the *SVID* `−e`. Because the two sys- 7
tems differed, a mnemonic compromise was selected. 7

The `−a` option is described with some optional behavior because the *SVID* omits 7
session leaders, but BSD does not. 7

In an earlier draft, format specifiers appeared for priority and start time. The former was not defined adequately in POSIX.2 and was removed in deference to the defined system scheduling priority; the latter because elapsed time was considered to be more useful.

The following categories of information were omitted because they were too implementation-specific:

`STATE`	Process state (`R` for ''Run'',etc.)
`FLAGS`	Octal scheduling flags.
`ADDR`	Process address.
`WCHAN`	Wakeup channel or event being slept on.

In a new BSD version of `ps`, a `−O` option can be used to write all of the default information followed by additional format specifiers. This was not adopted by POSIX.2 because the default output is implementation-defined. Nevertheless, this is a useful option that should be reserved for that purpose. In the `−o` option for the POSIX.2 `ps`, the format is the concatenation of each `-o`. Therefore, the user can have an alias or function that defines the beginning of her desired format and add more fields to the end of the output in certain cases where that would be useful.

The format of the terminal name is unspecified, but the descriptions of `ps`, `talk`, 7
`who`, and `write` require that they all use the same format. 7

The `pcpu` field indicates that the CPU time available is determined in an 8
unspecified manner. This is because it is difficult to express an algorithm that is 8
useful across all possible machine architectures. Historical counterparts to this 8
value have attempted to show percentage of use in the recent past, such as the 8
preceding minute. Frequently, these values for all processed did not add up to 8
100 percent. Implementations are encouraged to provide data in this field to 8
users that will help them identify processes currently affecting the performance of 8
the system. 8

5.24 `renice` — Set system scheduling priorities of running processes

5.24.1 Synopsis

`renice` [`−n` *increment*] [`−g` | `−p` | `−u`] *ID* . . .

Obsolescent Versions :

`renice` *nice_value* [`−p`] *pid* . . . [`−g` *gid* . . .] [`−p` *pid* . . .] [`−u` *user* . . .]

`renice` *nice_value* `−g` *gid* . . . [`−g` *gid* . . .] [`−p` *pid* . . .] [`−u` *user* . . .]

`renice` *nice_value* `−u` *user* . . . [`−g` *gid* . . .] [`−p` *pid* . . .] [`−u` *user* . . .]

5.24.2 Description

The `renice` utility shall request that the system scheduling priorities (see 7
2.2.2.212) of one or more running processes be changed. When a process ID is 7
`renice`d (the default), the applicable processes are specified by their process IDs. 7
When a process group is `renice`d, the request applies to all processes in the pro- 7
cess group. 7

The system scheduling priority shall be bounded in an implementation-defined manner. If the requested *increment* (or *nice_value* in the obsolescent versions) would raise or lower the system scheduling priority of the executed utility beyond implementation-defined limits, then the limit whose value was exceeded shall be used.

When a user is `renice`d and {_POSIX_SAVED_IDS} (see 2.9.3 in POSIX.1 {8}) is
defined, the request applies to all processes whose saved set-user- ID matches the 7
user ID corresponding to the user. When a user is `renice`d and
{_POSIX_SAVED_IDS} is not defined, the request applies to all processes whose 7
real user ID matches the user ID corresponding to the user.

Regardless of which options are supplied or any other factor, `renice` shall not 7
alter the system scheduling priorities of any process unless the user requesting 7
such a change has appropriate privileges to do so for the specified process. If the 7
user lacks appropriate privileges to perform the requested action, the utility shall 7
return an error status. 7

On systems where {_POSIX_SAVED_IDS} is defined, the saved set-user- ID of the 7
user's process shall be checked instead of its effective user ID when `renice` 7
attempts to determine the user ID of the process in order to determine whether 7
the user has appropriate privileges. On systems where {_POSIX_SAVED_IDS} is 7
not defined, `renice` shall check the real or effective user ID when attempting to 7
determine the user ID of the process and the user ID of the requester as part of 7
the determination of whether the user has appropriate privileges to perform the 7
requested action. 7

5.24.3 Options

The `renice` utility shall conform to the utility argument syntax guidelines described in 2.10.2. The obsolescent version shall conform with the following exceptions:

— The first operand, *nice_value* , shall precede the options and can have the appearance of a multidigit option.

— The `−g`, `−p`, and `−u` options can each take multiple option-arguments.

— The *pid* option-argument can be used without its `−p` option.

The following options shall be supported by the implementation:

`−g` Interpret all operands (or just the *gid* arguments in the obsolescent version) as unsigned decimal integer process group IDs.

`−n` *increment*

Specify how the system scheduling priority of the specified process(es) shall be adjusted. The *increment* option-argument is a positive or negative decimal integer that shall be used to modify the system scheduling priority of the specified process(es).

Positive *increment* values shall cause a lower system scheduling priority. Negative *increment* values may require appropriate privileges and shall cause a higher system scheduling priority.

`−p` Interpret all operands (or just the *pid* arguments in the obsolescent version) as unsigned decimal integer process IDs. The `−p` option is the default if no options are specified.

`−u` Interpret all operands (or just the *user* arguments in the obsolescent version) as users. If a user exists with a user name equal to the operand, then the user ID of that user shall be used in further processing. Otherwise, if the operand represents an unsigned decimal integer, it shall be used as the numeric user ID of the user.

5.24.4 Operands

The following operands shall be supported by the implementation:

ID A process ID, process group ID, or user name/user ID, depending on the option selected.

nice_value (Obsolescent.) The value specified shall be taken as the actual system scheduling priority, rather than as an increment to the existing system scheduling priority. Specifying a scheduling priority higher than that of the existing process may require appropriate privileges.

5.24.5 External Influences

5.24.5.1 Standard Input

None.

5.24.5.2 Input Files

None.

5.24.5.3 Environment Variables

The following environment variables shall affect the execution of `renice`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.24.5.4 Asynchronous Events

Default.

5.24.6 External Effects

5.24.6.1 Standard Output

None.

5.24.6.2 Standard Error

Used only for diagnostic messages.

5.24.6.3 Output Files

None.

5.24.7 Extended Description

None.

5.24.8 Exit Status

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

0 Successful completion.

>0 An error occurred.

5.24.9 Consequences of Errors

Default.

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

Usage, Examples

The *gid*, *pid*, and *user* specifications do not fit either the definition of operand or option-argument. However, for clarity, they have been included in the Options subclause, rather than the Operand subclause.

The following example would adjust the system scheduling priority in such a way that process IDs 987 and 32 would have a lower scheduling priority:

```
renice -n 5 -p 987 32
```

The next example would adjust the system scheduling priority in such a way that group IDs 324 and 76 would have a higher scheduling priority, if the user has the appropriate privileges to do so:

```
renice -n -4 -g 324 76
```

The next example would adjust the system scheduling priority in such a way that numeric user ID 8 and user `sas` would have a lower scheduling priority:

```
renice -n 4 -u 8 sas
```

Useful nice values on historical systems include 19 or 20 (the affected processes will run only when nothing else in the system attempts to run), 0 (the ‘‘base’’ scheduling priority), and any negative number (to make processes run faster).

History of Decisions Made

Originally this utility was written in the historical manner, using the term ‘‘nice value.’’ This was always a point of concern with users, because it was never intuitively obvious what this meant. With the new version of `renice`, which uses the term ‘‘system scheduling priority,’’ it is hoped that novice users will better understand what this utility was meant to do. It also made it much easier to document what the utility was meant to do.

The obsolescent version of `renice` allows pids, gids, and users to be mixed on the command line. This was seen as overly confusing, so when `renice` was rewritten, it was decided to allow only one kind of ID for each invocation. If no `−p`, `−g` or `−u` is specified, any ID operands are considered process IDs.

Strong input was received that the traditional syntax was very difficult to understand and it should be adjusted to comply with the utility syntax guidelines and be more reminiscent of the `nice` utility syntax.

Although this utility has use by system administrators (and in fact appears in the system administration portion of the BSD documentation), the developers of the standard felt that it was very useful for individual end users to control their own processes.

5.25 `split` — Split files into pieces

5.25.1 Synopsis

`split` [`−l` *line_count*] [`−a` *suffix_length*] [*file* [*name*]]

`split` `−b` *n*[`k`|`m`] [`−a` *suffix_length*] [*file* [*name*]]

Obsolescent Version :

`split` [`−`*line_count*] [`−a` *suffix_length*] [*file* [*name*]]

5.25.2 Description

The `split` utility shall read an input file and write one or more output files. The default size of each output file shall be 1000 lines. The size of the output files can be modified by specification of the `−b` or `−l` options. Each output file shall be created with a unique suffix. The suffix shall consist of exactly *suffix_length* lowercase letters from the POSIX Locale. The letters of the suffix shall be used as if they were a base-26 digit system, with the first suffix to be created consisting of all '`a`' characters, the second with a '`b`' replacing the last '`a`', etc., until a name of all '`z`'s is created. By default, the names of the output files shall be '`x`', followed by a two-character suffix from the character set as described above, starting with `aa`, `ab`, `ac`, etc., and continuing until the suffix `zz`, for a maximum of 676 files.

If the number of files required exceeds the maximum allowed by the suffix length
provided, such that the last allowable file would be larger than the requested size,
the `split` utility shall fail after creating the last file with a valid suffix; `split` 7
shall not delete the files it created with valid suffixes. If the file limit is not 7
exceeded, the last file created shall contain the remainder of the input file, and
may be smaller than the requested size.

5.25.3 Options

The `split` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that the obsolescent version allows a multidigit option, −*line_count* .

The following options shall be supported by the implementation:

`−a` *suffix_length*
: Use *suffix_length* letters to form the suffix portion of the filenames of the split file. If `−a` is not specified, the default suffix length shall be two. If the sum of the *name* operand and the *suffix_length* option-argument would create a filename exceeding {NAME_MAX} bytes, an error shall result; `split` shall exit with a diagnostic message and no files shall be created.

`-b` *n* Split a file into pieces *n* bytes in size.

`-b` *n*`k` Split a file into pieces *n*∗1024 bytes in size.

`-b` *n*`m` Split a file into pieces *n*∗1 048 576 bytes in size.

`-l` *line_count*
–*line_count* (Obsolescent.)
Specify the number of lines in each resulting file piece. The *line_count* argument is an unsigned decimal integer. The default is 1000. If the input does not end with a `<newline>`, the partial line shall be included in the last output file.

5.25.4 Operands

The following operands shall be supported by the implementation:

file The pathname of the ordinary file to be split. If no input file is given or *file* is –, the standard input shall be used.

name The prefix to be used for each of the files resulting from the split operation. If no *name* argument is given, '`x`'' shall be used as the prefix of the output files. The combined length of the basename of *prefix* and *suffix_length* cannot exceed {NAME_MAX} bytes; see 5.25.3.

5.25.5 External Influences

5.25.5.1 Standard Input

See Input Files.

5.25.5.2 Input Files

Any file can be used as input.

5.25.5.3 Environment Variables

The following environment variables shall affect the execution of `split`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.25.5.4 Asynchronous Events

Default.

5.25.6 External Effects

5.25.6.1 Standard Output

None.

5.25.6.2 Standard Error

Used only for diagnostic messages.

5.25.6.3 Output Files

The output files contain portions of the original input file, otherwise unchanged.

5.25.7 Extended Description

None.

5.25.8 Exit Status

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

0 Successful completion.

>0 An error occurred.

5.25.9 Consequences of Errors

Default.

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

Usage, Examples

In the following examples `foo` is a text file that contains 5000 lines.

This example creates five files, `xaa`, `xab`, `xac`, `xad`, and `xae`:

```
split foo
```

This example also creates five files, but the suffixed portion of the created files consists of three letters, `xaaa`, `xaab`, `xaac`, `xaad`, and `xaae`:

```
split -a 3 foo
```

This example creates three files with four-letter suffixes and a supplied prefix, `bar_aaaa`, `bar_aaab`, and `bar_aaac`:

```
split -a 4 -l 2000 foo bar_
```

This example creates as many files as are necessary to contain at most 20∗1024 bytes, each with the default prefix of ‘`x`’’ and a five-letter suffix:

```
split -a 5 -b 20k foo
```

History of Decisions Made

The −`b` option was added to provide a mechanism for splitting files other than by lines. While most uses of the −`b` option will be for transmitting files over networks, some felt it would have additional uses.

The −`a` option was added to overcome the limitation of being able to create only 676 files.

The −*line_count* option was declared obsolescent and replaced with the −`l` option. This allows future standards to have a `split` utility that fully conforms to the syntax guidelines.

Consideration was given to deleting this utility using the rationale that the function provided by this utility is available via the `csplit` utility (see 5.6). Upon reconsideration of the purpose of the User Portability Extension, it was decided to retain both this utility and the `csplit` utility because users use both utilities and have historical expectations of their behavior. Furthermore, the splitting on byte boundaries in `split` cannot be duplicated with the historical `csplit`.

The text ‘`split` shall not delete the files it created with valid suffixes’’ would nor- 7
mally be assumed, but since the related utility, `csplit`, does delete files under 7
some circumstances, the historical behavior of `split` is made explicit to avoid 7
misinterpretation. 7

5.26 `strings` — Find printable strings in files

5.26.1 Synopsis

`strings` [−a][−t *format*][−n *number*][*file* . . .]

Obsolescent Version :

`strings` [−][−t *format*][−*number*][*file* . . .]

5.26.2 Description

The `strings` utility shall look for printable strings in regular files and write those strings to standard output. A printable string is any sequence of four (by default) or more printable characters terminated by a `<newline>` or NUL character. Additional implementation-defined strings may be written. (See `localedef` in 4.35.)

5.26.3 Options

The `strings` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that the obsolescent version uses − in a nonstandard way and allows a multidigit option, −*number* .

The following options shall be supported by the implementation:

`−a`
`-` (Obsolescent.)
: Scan files in their entirety. If `−a` is not specified, it is implementation defined what portion of each file is scanned for strings.

`−n` *number*
−*number* (Obsolescent.)
: Specify the minimum string length, where the *number* argument is a positive decimal integer. The default shall be 4.

`−t` *format*
: Write each string preceded by its byte offset from the start of the file. The format shall be dependent on the single character used as the *format* option-argument:

- `d` The offset shall be written in decimal.
- `o` The offset shall be written in octal.
- `x` The offset shall be written in hexadecimal.

5.26.4 Operands

The following operand shall be supported by the implementation:

file A pathname of a regular file to be used as input. If no *file* operand is specified, the `strings` utility shall read from the standard input.

5.26.5 External Influences

5.26.5.1 Standard Input

See Input Files.

5.26.5.2 Input Files

The input files named by the utility arguments or the standard input shall be regular files of any format.

5.26.5.3 Environment Variables

The following environment variables shall affect the execution of `strings`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files), and to identify printable strings.

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.26.5.4 Asynchronous Events

Default.

5.26.6 External Effects

5.26.6.1 Standard Output

Strings found shall be written to the standard output, one per line.

When the `-t` option is not specified, the format of the output shall be:

`"%s"`, *<string>*

With the `-t o` option, the format of the output shall be:

`"%o %s"`, *<byte offset>*, *<string>*

With the `-t x` option, the format of the output shall be:

`"%x %s"`, *<byte offset>*, *<string>*

With the `-t d` option, the format of the output shall be:

`"%d %s"`, *<byte offset>*, *<string>*

5.26.6.2 Standard Error

Used only for diagnostic messages.

5.26.6.3 Output Files

None.

5.26.7 Extended Description

None.

5.26.8 Exit Status

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

- 0 Successful completion.
- >0 An error occurred.

5.26.9 Consequences of Errors

Default.

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

Usage, Examples

The `strings` utility is useful for identifying random object files and many other things.

History of Decisions Made

Apart from rationalizing the option syntax and slight difficulties with object and executable binary files, `strings` is specified to closely match existing practice. The `−a` and `−n` options were introduced to replace the nonconforming `−` and −*number* options.

Investigation revealed that `−o` historically means different things on different implementations. Some maintain that it means ''*offset* in decimal,'' while others maintain that it means ''offset in *octal*.'' Instead of trying to decide which way would be least objectionable, the `−t` option was added. It was originally named `−O` to mean ''offset,'' but was changed to `−t` to be consistent with `od`.

By default, the data area (as opposed to the text, ''bss,'' or header areas) of a binary executable file is scanned in historical implementations. Implementations are required to document which areas are scanned.

Some historical implementations do not require NUL or `<newline>` terminators for strings to permit those languages that do not use NUL as a string terminator to have their strings written.

The C Standard {7} function *isprint*() is restricted to a domain of *unsigned char*. This standard requires implementations to write strings as defined by the current locale.

5.27 `tabs` — Set terminal tabs

5.27.1 Synopsis

`tabs` [−*n*] [−`T` *type*]

`tabs` [−`T` *type*] *n1* [`,`*n2*`,`. . .]

5.27.2 Description

The `tabs` utility shall display a series of characters that first clears the hardware terminal tab settings and then initializes the tab stops at the specified positions.

The phrase ‘‘tab-stop position *N*’’ shall be taken to mean that, from the start of a line of output, tabbing to position *N* shall cause the next character output to be in the (N+1)th column position on that line. The maximum number of tab stops allowed is terminal dependent.

It need not be possible to implement `tabs` on certain terminals. If the terminal type obtained from the **TERM** environment variable or −`T` option represents such a terminal, an appropriate diagnostic message shall be written to standard error and `tabs` shall exit with a status greater than zero.

5.27.3 Options

The `tabs` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

−*n* Specify repetitive tab stops separated by a uniform number of column positions, *n*, where *n* is a single-digit decimal number. The default usage of `tabs` with no arguments shall be equivalent to `tabs −8`. When `−0` is used, the tab stops shall be cleared and no new ones set.

−`T` *type* Indicate the type of terminal. If this option is not supplied and
the **TERM** variable is unset or null, an unspecified default termi- 8
nal type shall be used. The setting of *type* shall take precedence 8
over the value in **TERM** .

5.27.4 Operands

The following operand shall be supported by the implementation:

n1 [`,`*n2*`,`. . .]
A single command-line argument that consists of tab-stop values separated using either commas or `<blank>`s. The tab-stop values shall be positive decimal integers in strictly ascending order. If

any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. For example, the tab lists `1,10,20,30` and `1,10,+10,+10` are considered to be identical.

5.27.5 External Influences

5.27.5.1 Standard Input

None.

5.27.5.2 Input Files

None.

5.27.5.3 Environment Variables

The following environment variables shall affect the execution of `tabs`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

TERM This variable shall be interpreted as the name of the terminal type. If this variable is unset or null, and if the `-T` 7
option is not specified, an unspecified default terminal 7
type shall be used. 7

5.27.5.4 Asynchronous Events

Default.

5.27.6 External Effects

5.27.6.1 Standard Output

If standard output is a terminal, the appropriate sequence to clear and set the tab stops may be written to standard output in an unspecified format. If standard output is not a terminal, undefined results occur.

5.27.6.2 Standard Error

Used only for diagnostic messages.

5.27.6.3 Output Files

None.

5.27.7 Extended Description

None.

5.27.8 Exit Status

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

0 Successful completion.

>0 An error occurred.

5.27.9 Consequences of Errors

Default.

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

Usage, Examples

Some integrated display units might not have escape sequences to set tab stops, but may be set by internal system calls. On these terminals, `tabs` will work if standard output is directed to the terminal; if output is directed to another file, however, `tabs` will fail.

History of Decisions Made

Consideration was given to having the `tput` utility handle all of the functions described here. However, the separate `tabs` utility was retained because it seems more intuitive to use a command named `tabs` than `tput` with a new option. The POSIX. 2a `tput` does not support setting or clearing tabs and no known historical version of `tabs` supports the capability of setting arbitrary tab stops.

The System V `tabs` interface is very complex; the version in this standard has a reduced feature list. There was considerable sentiment for specifying only a means of resetting the tabs back to a known state—presumably the ''standard'' tabs every eight positions. The following features were omitted:

(1) Setting tab stops tailored for certain programming languages; the working group was afraid that there would be no end to the bickering about which languages to include and where the tabs should be (even C programmers cannot agree on this).

(2) Setting tab stops via the first line in a file, using `--`*file*. Since even the *SVID* has no complete explanation of this feature, it is doubtful that it is in widespread use.

(3) Setting the left margin using `"+m`*n*`"`. As this does not work with all terminal types, it was omitted.

In an earlier draft, a `-t` *tablist* option was added for consistency with `expand`; this was later removed when inconsistencies with the historical list of tabs were identified.

Consideration was given to adding a `-p` option that would output the current tab settings so that they could be saved and then later restored. This was not accepted because querying the terminal's tab stops is not a capability in historical *terminfo* or *termcap* facilities and might not be supportable on a wide range of terminals.

5.28 `talk` — Talk to another user

5.28.1 Synopsis

`talk` *address* [*terminal*]

5.28.2 Description

The `talk` utility is a two-way, screen-oriented communication program.

When first invoked, `talk` shall send a message similar to:

```
Message from <unspecified string>
talk: connection requested by your_address
talk: respond with: talk your_address
```

to the specified *address* . At this point, the recipient of the message can reply by typing:

```
talk your_address
```

Once communication is established, the two parties can type simultaneously, with
their output displayed in separate regions of the screen. Characters shall be pro- 8
cessed as follows: 8

— Typing the `<alert>` character shall alert the recipient's terminal.

— Typing `control-L` shall cause the sender's screen regions to be refreshed.

— Typing the erase and kill characters shall affect the sender's terminal in the manner described by the POSIX.1 {8} *termios* interface.

— Typing the interrupt or end-of-file characters shall terminate the local `talk` utility. Once the `talk` session has been terminated on one side, the other side of the `talk` session shall be notified that the `talk` session has been terminated and shall be able to do nothing except exit.

— Typing characters from LC_CTYPE classifications `print` or `space` shall 8
cause those characters to be sent to the recipient's terminal.

— When and only when the `stty` *iexten* local mode (see 4.59.4.4) is enabled, 8
additional special control characters and multibyte or single-byte functions 8
shall be processed in an implementation-defined manner. (See POSIX.1 {8} 8
section 7.1.1.9.) 8

— Typing other nonprintable characters shall cause implementation-defined sequences of printable characters to be sent to the recipient's terminal.

Permission to be a recipient of a `talk` message can be denied or granted by use of
the `mesg` utility (see 5.17). However, a user's privilege may further constrain the 7
domain of accessibility of other users' terminals. The `talk` utility shall fail when 7
the user lacks the appropriate privileges to perform the requested action. 7

Certain block-mode terminals do not have all the capabilities necessary to support the simultaneous exchange of messages required for `talk`. When this type of exchange cannot be supported on such terminals, the implementation may support an exchange with reduced levels of simultaneous interaction or it may report an error describing the terminal-related deficiency.

5.28.3 Options

None.

5.28.4 Operands

The following operands shall be supported by the implementation:

address The recipient of the `talk` session. One form of *address* is the *<user name>*, as returned by the `who` utility (see 5.36). Other address formats and how they are handled are unspecified.

terminal If the recipient is logged in more than once, the *terminal* argument can be used to indicate the appropriate terminal name. If *terminal* is not specified, the `talk` message shall be displayed on one or more accessible terminals in use by the recipient. The format of *terminal* shall be the same as that returned by the `who` utility.

5.28.5 External Influences

5.28.5.1 Standard Input

Characters read from standard input shall be copied to the recipient's terminal in an unspecified manner. If standard input is not a terminal, talk shall write a diagnostic message and exit with a nonzero status.

5.28.5.2 Input Files

None.

5.28.5.3 Environment Variables

The following environment variables shall affect the execution of `talk`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files). If the recipient's locale does not use an LC_CTYPE equivalent to the sender's, the results are undefined.

LC_MESSAGES This variable shall determine the language in which diagnostic messages should be displayed.

TERM This variable shall be interpreted as the name of the invoker's terminal type. If this variable is unset or null, 7
an unspecified default terminal type shall be used. 7

5.28.5.4 Asynchronous Events

When the `talk` utility receives a SIGINT signal, the utility shall terminate and exit with a zero status. It shall take the standard action for all other signals; see 2.11.5.4.

5.28.6 External Effects

5.28.6.1 Standard Output

If standard output is a terminal, characters copied from the recipient's standard input may be written to standard output. Standard output also may be used for diagnostic messages. If standard output is not a terminal, `talk` shall exit with a nonzero status.

5.28.6.2 Standard Error

None.

5.28.6.3 Output Files

None.

5.28.7 Extended Description

None.

5.28.8 Exit Status

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

- 0 Successful completion.
- >0 An error occurred, or `talk` was invoked on a terminal incapable of supporting it.

5.28.9 Consequences of Errors

Default.

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

Usage, Examples

The handling of nonprintable characters is partially implementation defined because the details of mapping them to printable sequences is not needed by the user. Historical implementations, for security reasons, disallow the transmission
of nonprintable characters that may send commands to the other terminal. 8
Because the handling of nonprintable, nonspace characters is tied to the `stty` 8
description of *iexten*, implementation extensions within the terminal driver can be 8
accessed. For example, some implementations provide line editing functions with 8
certain control character sequences. 8

History of Decisions Made

The `write` utility was included in the standard since it can be implemented on all terminal types. The `talk` utility, which cannot be implemented on certain terminals, was considered to be a ‘‘better’’ communications interface. Both of these programs are in widespread use on historical implementations. Therefore, both utilities have been specified.

All references to networking abilities (`talk`ing to a user on another system) were removed as being outside the scope of the document.

The version of `talk` released with 4.3BSD uses a protocol that is incompatible with the protocol used in the version released with 4.2BSD.

Historical BSD and System V versions of `talk` terminate both of the conversations when either user breaks out of the session. This can lead to adverse consequences if a user unwittingly continues to enter text that will be interpreted by the shell when the other terminates the session. Therefore, the version of `talk` specified by this standard requires both users to explicitly terminate their end of the session.

Only messages sent to the user's own terminal can be internationalized in any way.

— The original ‘‘Message from <*unspecified string*> . . . ’’ message sent to the recipient’s terminal cannot be internationalized because the recipient’s environment is as yet inaccessible to the `talk` utility. The invoking party’s environment is irrelevant.

— Subsequent communication between the two parties cannot be internationalized because the two parties may specify different languages in their
environment (and nonportable characters cannot be mapped from one 7
language to another).

— Neither party can be required to communicate in a language other than C and/or the one specified by their environment because unavailable terminal hardware support (e.g., fonts) may be required.

The text in Standard Output reflects the usage of the verb ‘‘display’’ in this clause; some `talk` implementations actually use standard output to write to the terminal, but POSIX.2 does not require that to be the case.

The format of the terminal name is unspecified, but the descriptions of `ps`, `talk`, 7
`who`, and `write` require that they all use or accept the same format. 7

5.29 `time` — Time a simple command

5.29.1 Synopsis

`time [−p]` *utility* `[`*argument* ... `]`

5.29.2 Description

The `time` utility shall invoke the utility named by the *utility* operand with arguments supplied as the *argument* operands and write a message to standard error that lists timing statistics for the utility. The message shall include the following information:

— The elapsed (''real'') time between invocation of *utility* and its termination.

— The User CPU time, equivalent to the sum of the *tms_utime* and *tms_cutime* fields returned by the POSIX.1 {8} *times*() function for the process in which *utility* is executed.

— The System CPU time, equivalent to the sum of the *tms_stime* and *tms_cstime* fields returned by the POSIX.1 {8} *times*() function for the process in which *utility* is executed.

The precision of the timing shall be no less than the granularity defined for the size of the clock tick unit on the system, but the results shall be reported in terms of standard time units (e.g., 0.02 seconds, 00:00:00.02, 1m33.75s, 365.21 seconds), not numbers of clock ticks.

When `time` is used as part of a pipeline, the times reported are unspecified, 8
except when it is the sole command within a grouping command (see 3.9.4.1) in 8
that pipeline. For example, the commands on the left are unspecified; those on 8
the right report on utilities `a` and `c`, respectively. 8

```
time a | b | c          { time a } | b | c          8
a | b | time c          a | b | (time c)            8
```

5.29.3 Options

The `time` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following option shall be supported by the implementation:

`−p` Write the timing output to standard error in the format shown in 7
5.29.6.2. 7

5.29.4 Operands

The following operands shall be supported by the implementation:

utility The name of a utility that is to be invoked. If the *utility* operand names any of the special built-in utilities in 3.14, the results are undefined.

argument Any string to be supplied as an argument when invoking the utility named by the *utility* operand.

5.29.5 External Influences

5.29.5.1 Standard Input

None.

5.29.5.2 Input Files

None.

5.29.5.3 Environment Variables

The following environment variables shall affect the execution of `time`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

LC_NUMERIC This variable shall determine the locale for numeric formatting.

PATH This variable shall determine the search path that shall be used to locate the utility to be invoked. See 2.6.

5.29.5.4 Asynchronous Events

Default.

5.29.6 External Effects

5.29.6.1 Standard Output

None.

5.29.6.2 Standard Error

The standard error shall be used to write the timing statistics described in 5.29.2. If `-p` is specified, the following format shall be used in the POSIX Locale:

`"real %f\nuser %f\nsys %f\n"`, *<real seconds>*, *<user seconds>*, *<system seconds>*

where each floating point number shall be expressed in seconds. The precision used may be less than the default six digits of `%f`, but shall be sufficiently precise to accommodate the size of the clock tick on the system (for example, if there were 60 clock ticks per second, at least two digits shall follow the radix character). The number of digits following the radix character shall be no less than one, even if this always results in a trailing zero. The implementation may append white space and additional information following the format shown here.

5.29.6.3 Output Files

None.

5.29.7 Extended Description

None.

5.29.8 Exit Status

If the *utility* utility is invoked, the exit status of `time` shall be the exit status of *utility;* otherwise, the `time` utility shall exit with one of the following values: 7

1–125 An error occurred in the `time` utility. 7

126 The utility specified by *utility* was found but could not be invoked. 7

127 The utility specified by *utility* could not be found. 7

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

Examples, Usage

It is frequently desirable to apply `time` to pipelines or lists of commands. This can be done by placing pipelines and command lists in a single file; this file can then be invoked as a utility, and the `time` applies to everything in the file.

Alternatively, the following command can be used to apply `time` to a complex command:

`time sh -c '`*complex-command-line*`'`

However, note that on some historical systems that do not implement job control, 8
not all of the processes in a pipeline have to be owned by the same parent process 8
(they are sometimes owned by `init`). Some of the process times in a pipeline are 8
therefore invisible to the POSIX.1 {8} *times*() function and to the `time` utility. 8

The `command`, `env`, `nice`, `nohup`, `time`, and `xargs` utilities have been specified to 7
use exit code 127 if an error occurs so that applications can distinguish ''failure to 7
find a utility'' from ''invoked utility exited with an error indication.'' The value 7
127 was chosen because it is not commonly used for other meanings; most utilities 7
use small values for ''normal error conditions'' and the values above 128 can be 7
confused with termination due to receipt of a signal. The value 126 was chosen in 7
a similar manner to indicate that the utility could be found, but not invoked. 7
Some scripts produce meaningful error messages differentiating the 126 and 127 7
cases. The distinction between exit codes 126 and 127 is based on KornShell 8
practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], 8
and uses 126 when any attempt to *exec* the utility fails for any other reason. 8

History of Decisions Made

The `time` utility was originally proposed for the base POSIX.2 standard, but it was rejected because it was not useful for portable applications:

— The underlying CPU definitions from POSIX.1 {8} are somewhat vague, so the numeric output could not be compared accurately between systems, or even between invocations.

— The creation of portable benchmark programs was outside the POSIX.2 scope of work.

However, `time` does fit in the scope of user portability. Human judgment can be applied to the analysis of the output and it could be very useful in hands-on debugging of applications or in providing subjective measures of system performance.

The default output format has been left unspecified because historical implementations differ greatly in their style of depicting the numbers. The `-p` was invented to provide scripts a common means of obtaining this information.

The KornShell `time` is shell reserved word that can be used to time an entire 8
pipeline, rather than just a simple command. The POSIX definition has been 8

worded to allow this implementation. Consideration was given to invalidating this approach because of the historical model from the C-shell and System V shell. However, since the System V `time` historically has not produced accurate results in pipeline timing (because the constituent processes are not all owned by the same parent process, as allowed by POSIX), it did not seem worthwhile to break existing KornShell usage.

The term *utility* is used, rather than *command* , to highlight the fact that shell compound commands, pipelines, special built-ins, etc., cannot be used directly. However, *utility* includes user application programs and shell scripts, not just the standard utilities.

5.30 `tput` — Change terminal characteristics

5.30.1 Synopsis

`tput [−T` *type* `]` *operand* . . .

5.30.2 Description

The `tput` utility shall display terminal-dependent information. The manner in which this information is retrieved is unspecified. The information displayed shall clear the terminal screen, initialize the user's terminal, or reset the user's terminal, depending on the operand given. The exact consequences of displaying this information are unspecified.

5.30.3 Options

The `tput` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following option shall be supported by the implementation:

`−T` *type* Indicate the type of terminal. If this option is not supplied and the **TERM** variable is unset or null, an unspecified default terminal type shall be used. The setting of *type* shall take precedence over the value in **TERM** .

5.30.4 Operands

The following strings shall be supported as operands by the implementation in the POSIX Locale:

`clear` Display the clear-screen sequence.

`init` Display the sequence that will initialize the user's terminal in an implementation-defined manner.

`reset` Display the sequence that will reset the user's terminal in an implementation-defined manner.

If a terminal does not support any of the operations described by these operands, this shall not be considered an error condition.

5.30.5 External Influences

5.30.5.1 Standard Input

None.

5.30.5.2 Input Files

None.

5.30.5.3 Environment Variables

The following environment variables shall affect the execution of `tput`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be displayed.

TERM This variable shall be interpreted as the name of the terminal type. If this variable is unset or null, and if the `−T` 7
option is not specified, an unspecified default terminal 7
type shall be used. 7

5.30.5.4 Asynchronous Events

Default.

5.30.6 External Effects

5.30.6.1 Standard Output

If standard output is a terminal device, it may be used for writing the appropriate sequence to clear the screen or reset or initialize the terminal. If standard output is not a terminal device, undefined results occur.

5.30.6.2 Standard Error

Used only for diagnostic messages.

5.30.6.3 Output Files

None.

5.30.7 Extended Description

None.

5.30.8 Exit Status

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

- 0 The requested string was written successfully.
- 1 Unspecified.
- 2 Usage error.
- 3 No information is available about the specified terminal type.
- 4 The specified operand is invalid.
- >4 An error occurred.

5.30.9 Consequences of Errors

If one of the operands is not available for the terminal, `tput` continues processing the remaining operands.

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

Usage, Examples

Initialize the terminal according to the type of terminal in the environmental variable **TERM**. This command can be included in a `.profile` file.

```
tput init
```

Reset a 450 terminal.

```
tput -T 450 reset
```

Some terminals use control characters to perform the stated functions, and on such terminals it might make sense to use `tput` to store the initialization strings in a file or environment variable for later use. However, because other terminals might rely on system calls to do this work, the standard output cannot be used in a portable manner, such as the following nonportable constructs:

```
ClearVar=‘tput clear‘
tput reset | mailx -s "Wake Up" ddg
```

History of Decisions Made

The list of operands was pared down to a bare minimum for the following reasons:

(1) The only features chosen were those that were likely to be used by human users interacting with a terminal.

(2) Specifying the full *terminfo* set was not considered desirable, but the working group did not want to get into a ‘‘my favorite operand is ...’’ dispute with balloters.

(3) This standard is not attempting to provide applications with fancy terminal handling capabilities, as that falls outside of its assigned scope, and intersects with the responsibilities of other standards bodies.

The difference between resetting and initializing a terminal is left unspecified, as they vary greatly based on hardware types. In general, resetting is a more severe action.

Exit status of 1 is historically reserved for finding out if a Boolean operand is not set. Although the operands were pared down to a minimum set, the exit status of 1 should still be reserved for the Boolean operands, for those sites that wish to support them.

5.31 `unalias` — Remove alias definitions

5.31.1 Synopsis

```
unalias alias-name ...
unalias −a
```

5.31.2 Description

The `unalias` utility shall remove the definition for each alias name specified. See 3.3.1. The aliases shall be removed from the current shell execution environment; see 3.12.

5.31.3 Options

The `unalias` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following option shall be supported by the implementation:

`−a` Remove all alias definitions from the current shell execution environment.

5.31.4 Operands

The following operand shall be supported by the implementation:

alias-name The name of an alias to be removed.

5.31.5 External Influences

5.31.5.1 Standard Input

None.

5.31.5.2 Input Files

None.

5.31.5.3 Environment Variables

The following environment variables shall affect the execution of `unalias`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.31.5.4 Asynchronous Events

Default.

5.31.6 External Effects

5.31.6.1 Standard Output

None.

5.31.6.2 Standard Error

Used only for diagnostic messages.

5.31.6.3 Output Files

None.

5.31.7 Extended Description

None.

5.31.8 Exit Status

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

0 Successful completion.

>0 One of the *alias-name* operands specified did not represent a valid alias definition, or an error occurred.

5.31.9 Consequences of Errors

Default.

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

Usage, Examples

None.

History of Decisions Made

The `unalias` description is based on that from historical KornShell implementations. Known differences exist between that and the C-Shell. The KornShell version was adopted to be consistent with all the other KornShell features in POSIX.2, such as command-line editing.

Since `unalias` affects the current shell execution environment, it is generally provided as a shell regular built-in.

The `-a` option is the equivalent of the C-shell's `unalias *` form and is provided to address security concerns about unknown aliases entering a user's (or application's) environment through the allowable implementation-provided
predefined alias route or as a result of an **ENV** file. (Although `unalias` could be 7
used to simplify the ''secure'' shell script shown in the `command` rationale, it does 7
not obviate the need to quote all command names. An initial call to `unalias -a` 7
would have to be quoted in case there was an alias for `unalias`.) 7

5.32 `unexpand` — Convert spaces to tabs

5.32.1 Synopsis

`unexpand` [`–a` | `–t` *tablist*] [*file* . . .]

5.32.2 Description

The `unexpand` utility shall copy files or standard input to standard output, converting `<blank>`s at the beginning of each line into the maximum number of `<tab>`s followed by the minimum number of `<space>`s needed to fill the same column positions originally filled by the translated `<blank>`s. By default, tabstops shall be set at every eighth column position. Each `<backspace>` character shall be copied to the output, and shall cause the column position count for tab calculations to be decremented; the count shall never be decremented to a value less than one.

5.32.3 Options

The `unexpand` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following option shall be supported by the implementation:

`–a` In addition to translating `<blank>`s at the beginning of each line, translate all sequences of two or more `<blank>`s immediately preceding a tab stop to the maximum number of `<tab>`s followed by the minimum number of `<space>`s needed to fill the same column positions originally filled by the translated `<blank>`s.

`–t` *tablist* Specify the tab stops. The option-argument *tablist* shall be a sin- 8
gle argument consisting of a single positive decimal integer or 8
multiple positive decimal integers, separated by `<blank>`s or commas, in ascending order. If a single number is given, tabs shall be set *tablist* column positions apart instead of the default 8. If multiple numbers are given, the tabs shall be set at those specific column positions.

Each tab-stop position *N* shall be an integer value greater than zero, and the list shall be in strictly ascending order. This shall be taken to mean that, from the start of a line of output, tabbing to position *N* shall cause the next character output to be in the (*N*+1)th column position on that line. When the `–t` option is not specified, the default shall be the equivalent of specifying `-t 8` (except for the interaction with `–a`, described below).

No `<space>`-to-`<tab>` conversions shall occur for characters at positions beyond the last of those specified in a multiple tab-stop list.

When `−t` is specified, the presence or absence of the `−a` option shall be ignored; conversion shall not be limited to the processing of leading `<blank>`s.

5.32.4 Operands

The following operand shall be supported by the implementation:

file A pathname of a text file to be used as input.

5.32.5 External Influences

5.32.5.1 Standard Input

See Input Files.

5.32.5.2 Input Files

The input files shall be text files.

5.32.5.3 Environment Variables

The following environment variables shall affect the execution of `unexpand`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files), the processing of `<tab>`s and `<space>`s, and for the determination of the width in column positions each character would occupy on a constant-width-font output device.

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.32.5.4 Asynchronous Events

Default.

5.32.6 External Effects

5.32.6.1 Standard Output

The standard output is equivalent to the input files with the specified `<space>` to `<tab>` conversions.

5.32.6.2 Standard Error

Used only for diagnostic messages.

5.32.6.3 Output Files

None.

5.32.7 Extended Description

None.

5.32.8 Exit Status

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

- 0 Successful completion.
- >0 An error occurred.

5.32.9 Consequences of Errors

Default.

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

Usage, Examples

None.

History of Decisions Made

On several occasions, consideration was given to adding a `−t` option to the `unexpand` utility to complement the `−t` in `expand` (see 5.11). The historical intent of `unexpand` was to translate multiple `<blank>`s into tab stops, where tab stops were a multiple of eight column positions on most UNIX systems. A previous draft omitted `−t` because it seemed outside the UPE scope; it was not described in any of the Base Documents. However, a good case was made that hard-coding tab stops every eight columns was not suitable for the international community and broke historical precedents for some vendors in the FORTRAN community, so `−t` was restored in conjunction with the list of valid extension categories considered by the working group. Thus, `unexpand` is now the logical converse of `expand`, as one might expect.

One nonintuitive aspect of `unexpand` is its restriction to leading spaces when `−a` (and now, `−t`) are not specified. This is historical practice and there is no compelling reason to change it. Users who desire converting all spaces in the file can easily alias `unexpand` to use the `−a` or `-t 8` option.

5.33 uudecode — Decode a binary file

5.33.1 Synopsis

`uudecode` [*file*]

5.33.2 Description

The `uudecode` utility shall read a file, or standard input if no file is specified, that includes data created by the `uuencode` utility (see 5.34). The `uudecode` utility shall scan the input file, searching for data compatible with the format specified in 5.34.6.1 and attempt to create or overwrite the file described by the data. The pathname, file access permission bits, and contents for the file to be produced shall all be contained in that data. The mode bits of the created file shall be set from the file access permission bits contained in the data; i.e., other attributes of the mode, including the file mode creation mask (see `umask` in 4.67), 7
shall not affect the file being produced. 7

If the pathname of the file to be produced exists, and the user does not have write permission on that file, `uudecode` shall terminate with an error. If the pathname of the file to be produced exists, and the user has write permission on that file, the existing file shall be overwritten.

If the input data was produced by `uuencode` on a system with a different number of bits per byte than on the target system, the results of `uudecode` are unspecified.

5.33.3 Options

None.

5.33.4 Operands

The following operand shall be supported by the implementation:

file The pathname of a file containing the output of `uuencode`; see 5.34.6.1.

5.33.5 External Influences

5.33.5.1 Standard Input

See Input Files.

5.33.5.2 Input Files

The input files shall be files containing the output of `uuencode`; see 5.34.6.1.

5.33.5.3 Environment Variables

The following environment variables shall affect the execution of `uudecode`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g.,
single- versus multibyte characters in arguments and 8
input files). 8

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.33.5.4 Asynchronous Events

Default.

5.33.6 External Effects

5.33.6.1 Standard Output

None.

5.33.6.2 Standard Error

Used only for diagnostic messages.

5.33.6.3 Output Files

The output file shall be in the same format as the file originally encoded by `uuencode`.

5.33.7 Extended Description

None.

5.33.8 Exit Status

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

0	Successful completion.
>0	An error occurred.

5.33.9 Consequences of Errors

Default.

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

Usage, Examples

The user who is invoking `uudecode` must have write permission on the specified file.

History of Decisions Made

Input files are not necessarily text files, as stated by an earlier draft. Although the `uuencode` output is a text file, that output could have been wrapped within another file or mail message that is not a text file.

The output of `uuencode` is essentially an encoded bit stream that is not cognizant of byte boundaries. It is possible that a 9-bit byte target machine can process input from an 8-bit source, if it is aware of the requirement, but the reverse is unlikely to be satisfying. Of course, the only data that is meaningful for such a transfer between architectures is generally character data.

5.34 uuencode — Encode a binary file

5.34.1 Synopsis

`uuencode` [*file*] *decode_pathname*

5.34.2 Description

The `uuencode` utility shall write an encoded version of the named input file, or standard input if no *file* is specified, to standard output. The output shall be encoded using the algorithm described in 5.34.6.1, and shall include the file access permission bits (in `chmod` octal or symbolic notation; see 4.7) of the input file and the *decode_pathname* , for recreation of the file on another system that conforms to this standard.

5.34.3 Options

None.

5.34.4 Operands

The following operands shall be supported by the implementation:

decode_pathname
: The pathname of the file into which the `uudecode` utility (see 5.33) shall place the decoded file. If there are characters in *decode_pathname* that are not in the portable filename character set (see 2.2.2.111 in POSIX.1 {8}), the results are unspecified.

file
: A pathname of the file to be encoded.

5.34.5 External Influences

5.34.5.1 Standard Input

See Input Files.

5.34.5.2 Input Files

Input files can be files of any type.

5.34.5.3 Environment Variables

The following environment variables shall affect the execution of `uuencode`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and data).

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.34.5.4 Asynchronous Events

Default.

5.34.6 External Effects

5.34.6.1 Standard Output

The standard output shall be a text file (encoded in the character set of the current locale) that begins with the line:

`"begin∆%s∆%s\n"`, *<mode>*, *decode_pathname*

and ends with the line:

`"end\n"`

In both cases, the lines shall have no preceding or trailing `<blank>`s.

The algorithm that shall be used for lines in between `begin` and `end` takes three octets as input and writes four characters of output by splitting the input at six-bit intervals into four octets, containing data in the lower six bits only. These octets shall be converted to characters by adding a value of 0x20 to each octet, so that each octet is in the range 0x20–0x5f, and then it shall be assumed to represent a printable character in the ISO/IEC 646 {1} encoded character set. It then shall be translated into the corresponding character codes for the code set in use in the current locale. (For example, the octet 0x41, representing `A`, would be translated to `A` in the current code set, such as 0xc1 if it were EBCDIC.)

Where the bits of two octets are combined, the least significant bits of the first octet shall be shifted left and combined with the most significant bits of the second octet shifted right. Thus the three octets A, B, C are converted into the four octets:

```
0x20 + (( A >> 2                    ) & 0x3F)
0x20 + (((A << 4) | ((B >> 4) & 0xF)) & 0x3F)
0x20 + (((B << 2) | ((C >> 6) & 0x3)) & 0x3F)
0x20 + (( C                         ) & 0x3F)
```

These octets are then translated into the local character set.

Each encoded line shall contain a length character, equal to the number of characters to be decoded plus 0x20 translated to the local character set as described above, followed by the encoded characters. The maximum number of octets to be encoded on each line shall be 45.

5.34.6.2 Standard Error

Used only for diagnostic messages.

5.34.6.3 Output Files

None.

5.34.7 Extended Description

None.

5.34.8 Exit Status

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

0 Successful completion.

>0 An error occurred.

5.34.9 Consequences of Errors

Default.

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

Usage, Examples

The file is expanded by 35 percent (each three octets become four, plus control information) causing it to take longer to transmit.

History of Decisions Made

Since this utility is intended to create files to be used for data interchange between systems with possibly different code sets, and to represent binary data as a text file, ISO/IEC 646 {1} was chosen for a midpoint in the algorithm as a known reference point. The output from `uuencode` is a text file on the local system. If the output were in the ISO/IEC 646 {1} code set, it might not be a text file (at least because the `<newline>`s might not match), and the goal of creating a text file would be defeated. If this text file was then carried to another machine with the same code set, it would be perfectly compatible with that system's `uudecode`. If it was transmitted over a mail system or sent to a machine with a different code set, it is assumed that, as for every other text file, some translation mechanism would convert it (by the time it reached a user on the other system) into an appropriate code set. This translation only makes sense from the local code set, not if the file has been put into an ISO/IEC 646 {1} representation first. Similarly, `uuencode`d files can be placed in `pax` archives, intermixed with other text files in the same code set.

The algorithm is described in terms of 8-bit quantities, or octets. Since no byte alignment is implied, it will encode data from machines with any number of bits per byte. However, unless that encoded data is then decoded on a machine with the same number of bits per byte, the output might not be useful.

The term ''source file'' in previous drafts was changed to ''file'' to avoid any implied relationship to computer-language source code files.

5.35 `vi` — Screen-oriented (visual) display editor

This utility shall be provided on systems that both support the User Portability 7
Utilities Option and define the {POSIX2_CHAR_TERM} symbol. On other systems 7
it is optional. 7

5.35.1 Synopsis

`vi` [`−rR`][`−c` *command*][`−t` *tagstring*][`−w` *size*][*file* . . .]

Obsolescent Version :

`vi` [`−rR`][+*command*][`−t` *tagstring*][`−w` *size*][*file* . . .]

5.35.2 Description

The `vi` (visual) utility is a screen-oriented text editor. The user can switch back and forth between `vi` and the line editor `ex` (see 5.10) and execute `ex` commands from within `vi`.

When using `vi`, the terminal screen acts as a window into the editing buffer. Changes made to the editing buffer shall be reflected in the screen display; the position of the cursor on the screen shall indicate the position within the editing buffer.

Certain block-mode terminals do not have all the capabilities necessary to support the complete `vi` definition. When these commands cannot be supported on such terminals, this condition shall not produce an error message such as ‘‘not an editor command’’ or report a syntax error. The implementation may either accept the commands and produce results on the screen that are the result of an unsuccessful attempt to meet the requirements of this standard or report an error describing the terminal-related deficiency.

5.35.3 Options

The `vi` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except for the obsolescent +*command* ‘‘option.’’

The following options shall be supported by the implementation:

`−c` *command*

+*command* (Obsolescent.)

Begin editing by executing the specified `ex` command-mode command(s). As with normal `ex` command-line entries, the *command* option-argument can consist of multiple `ex` commands separated by vertical-line characters (`|`). The use of commands that enter input mode in this manner produces undefined results.

`−r` Attempt to recover the named *file*s after an editor or system crash, after the editor has been terminated by a signal or after the use of a `pre` (`ex`) editor command.

If no *file* operands are given, all other options, the **EXINIT** vari- 8
able, and any `.exrc` files shall be ignored; a list of all recoverable 8
files available to the invoking user shall be written; and `vi` shall 8
exit without reading files or processing user commands. 8

`−R` Set read-only mode, preventing accidental overwriting of the files. 8
Any command that would write to a file shall require the ! suffix 8
(see, for example, 5.10.7.2.37) to be effective in this mode. 8

`−t` *tagstring*
Edit the file containing the specified *tagstring* and set the initial position within the edit buffer to the point of definition of the tag. (See `ctags` in 5.7.) The tags feature represented by `−t` *tagstring*
and the `ta` command (see 5.10.7.2.32) is optional. It shall be pro- 8
vided on any system that also provides a conforming implementa- 8
tion of `ctags`; otherwise, the use of `−t` produces undefined 7
results. 7

`−w` *size* Set the value of the *window* editor option to *size*. See 5.10.7.5.28.

If both the `−t` *tagstring* and `−c` *command* (or the obsolescent +*command*) options 8
are given, the `−t` *tagstring* shall be processed first; i.e., the file containing the tag 8
is selected by `−t` and then the command is executed. 8

5.35.4 Operands 8

The following operand shall be supported by the implementation: 8

file A pathname of a file to be edited. 8

5.35.5 External Influences 8

5.35.5.1 Standard Input 8

The standard input consists of a series of commands and input text, as described 8
in 5.35.7. 8

5.35.5.2 Input Files 8

Input files shall be text files or files that would be text files except for an incom- 8
plete last line that is not longer than {LINE_MAX} − 1 bytes in length and contains 8
no NUL characters. The editing of other forms of files may optionally be allowed 8
by `vi` implementations. 7

The `.exrc` files (see 5.35.7) shall be text files consisting of `ex` commands. 8

By default, `vi` shall read lines from the files to be edited without interpreting any 8
of those lines as any form of editor command. 8

5.35.5.3 Environment Variables

The following environment variables shall affect the execution of `vi`:

COLUMNS This variable shall override the system-selected horizontal 7
screen size. See 2.6 for valid values and results when it is 7
unset or null. 7

EXINIT This variable shall be interpreted to contain a list of `ex` commands that shall be executed on editor startup, before reading the first file. The list can contain multiple commands by separating them using a vertical-line (`|`) character. See also 5.35.7 for more details of the initialization phase.

HOME This variable shall be interpreted as a pathname of a directory that shall be searched for an editor startup file named `.exrc`; see 5.35.7 for details.

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_COLLATE This variable shall determine the locale for character collation information in regular expressions.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files), the behavior of character classes within regular expressions, the classification of characters as upper- or lower-case letters, the case conversion of letters, and the detection of word boundaries.

LC_MESSAGES This variable shall determine the language in which messages should be displayed.

LINES This variable shall override the system-selected vertical 7
screen size, used as the number of lines in a screenful and 7
the vertical screen size in visual mode. See 2.6 for valid 7
values and results when it is unset or null. 7

PATH This variable shall determine the search path for the shell command specified in the editor commands `shell`, `read`, and `write` and the visual-mode command `!`; see the description of command search and execution in 3.9.1.1. 8

SHELL This variable shall be interpreted as the preferred command-line interpreter for use in `!`, `shell`, `read`, and other commands with an operand of the form `!`*string*. For the `shell` command the program shall be invoked with the single argument `−i`, for all others it shall be invoked with the two arguments `−c` and *string*. If this variable is null or not set, the `sh` (see 4.56) utility shall be used.

TERM This variable shall be interpreted as the name of the terminal type. If this variable is unset or null, an 7
unspecified default terminal type shall be used. 7

5.35.5.4 Asynchronous Events

The following actions shall be taken upon receipt of signals:

SIGINT The current editor command shall be aborted. The editor shall prompt for another command.

SIGCONT The screen shall be refreshed (if in *visual* mode).

SIGHUP If the current buffer has changed since the last `e` or `w` command, 8
`vi` shall attempt to save the current file in a state such that it can be recovered later by an `ex` or `vi −r` command.

The action taken for all other signals is unspecified. 7

5.35.6 External Effects

5.35.6.1 Standard Output

If standard output is a terminal device, it may be used for writing prompts to the user, for informational messages, and for writing lines from the file. If standard output is not a terminal device, undefined results occur.

5.35.6.2 Standard Error

Used only for diagnostic messages.

5.35.6.3 Output Files

The output from `vi` shall be text files that shall be identical to the input files if no changes have been made to those files by commands, with the exception that in all cases where a forced session termination (the `ex` command `q!`; see 5.10.7.2.23) has not been issued prior to any write of the file, a trailing `<newline>` shall be added to the last line of the file if one was not present in the input.

5.35.7 Extended Description

Only the visual mode of the editor is described in this subclause. See 5.10 for additional editing capabilities used in `vi`.

The pathname of the file being edited by `vi` is the *current* file. The text of the file shall be read into a *buffer*, and all editing changes shall be performed in this buffer; changes shall have no effect on the file until the buffer is written out explicitly. (The use of the term *buffer* does not attempt to imply a specific implementation.) Lines in the buffer may each be limited to {LINE_MAX} bytes and an error message may be displayed if the limit is exceeded during editing.

The *alternate* pathname is the name of the last file mentioned in an editor command, or the previous current pathname if the last file mentioned became the current file. When the character `%` appears in a pathname entered as part of a command argument, the character shall be replaced by the current pathname; the character `#` shall be replaced by the alternate pathname. The characters `%` and `#` can be escaped by preceding them with a backslash.

During initialization, before the first file is read or any user commands from the terminal are processed, if the environment variable **EXINIT** is set, the editor shall execute the `ex` commands contained in that variable. If the variable is not set, `vi` shall attempt to read `ex` commands from the file `$HOME/.exrc` (the file `.exrc` in the directory referred to by the **HOME** environment variable). If and only if **EXINIT** or `$HOME/.exrc` sets the editor option `exrc`, `vi` finally shall attempt to read `ex` commands from a file `.exrc` in the current directory. In the event that **EXINIT** is not set and the current directory is the user's home directory, any `.exrc` file shall only be processed once. No `.exrc` file shall be read unless it is owned by the same user ID as the effective user ID of the process. After any `.exrc` files are processed, any commands specified by the `−c` option shall be processed.

After initialization, `vi` shall be in *command* mode; *input* mode can be entered by several commands used to insert or change text. In input mode, `<ESC>` can be used to return to command mode; other uses of `<ESC>` are described in 5.35.7.1.17. The results of entering `<newline>`s in input mode are affected by the setting of the `autoindent` editor variable; see 5.10.7.5.1.

The last (bottom) line of the screen shall be used to display the input for search commands (`/` and `?`), for `ex` commands (`:`), and system commands (`!`). It shall be also used to report errors or display informational messages. The editor shall prompt for input for commands by displaying the command (`/`, `?`, `:`, or `!`) at the beginning of the last line of the screen. All subsequent characters shall be then taken as input until a line terminator is entered.

An interrupt typed during text input, or during the input of a command on the bottom line, shall terminate the input (or cancel the command) and shall return the editor to command mode. During command mode, an interrupt shall alert the terminal. In general, an alert indicates an error (such as unrecognized key).

Lines displayed on the screen containing only a tilde (~) shall indicate that the last line above them is the last line in the editing buffer (the ~ lines are past the end of the editing buffer).

There may be lines on the screen marked with an @. These indicate space on the screen not corresponding to lines in the file. This shall happen if there is space at the bottom of the screen for some text, but the next line of the buffer will not fit. (It may also happen on a terminal with limited local intelligence. In this case, these lines can be removed by entering a `control-R`, forcing the editor to refresh the screen.)

5.35.7.1 `vi` Command Descriptions

The cursor, in general, shall be placed on the current line and in the current column position as noted for each command described below. If the current line is not in the display window, then the display window shall be either scrolled or refreshed to cause the current line to be in the display window. If the screen is refreshed, the current line shall be positioned as close to the center of the display window as possible. If the current line is less than one-half window lines from the beginning of the editing buffer, the first line of the buffer shall be displayed on the first line of the display window. If the current line is less than one-half window lines from the end of the editing buffer, the remaining lines of the display window after the last line of the file shall contain only a ~ character in column position 1. If the screen is scrolled rather than refreshed, the current line shall be placed at the top of the display window if the current line is before the first line displayed. If the current line is after the last line displayed, the display window shall be scrolled up and the current line shall be placed on the last line of the display window.

As stated previously, the cursor is generally placed on the current column position of the current line. The one exception to this is if the current column position is beyond the end of the current line. In this case, the cursor shall be placed on the last character of the current line. However, the current column position value shall not be altered by this. Thus, if the current line changes to a longer line, the cursor shall be moved back out to the current column position or to the end of the line if it is shorter than the current column.

Unless otherwise specified, the commands shall be interpreted in command mode and have no special effect in input mode.

Some of the following command descriptions (such as `a`, `A`, `c`, etc.) move the cursor 7
and enter input mode. The indications of *Current line* and *Current column* are for 7
the cursor movement up to the beginning of input mode. Depending on the text 7
inserted, the cursor will generally wind up in a totally different location by the 7
time command mode is reentered. The cursor movement that accompanies the 7
transition from input mode to command mode is described in 5.35.7.1.17. 7

The following symbols are used in this section to represent arguments to commands, to describe commands, or to specify the new values of the current line and column indicators following execution of the command.

bigword A maximal sequence of nonblank characters preceded and followed by <blank> characters or the beginning or end of a line or the file.

buffer One of a number of named areas for saving text. Commands that change or delete text can be preceded by a buffer specification argument *buffer*. It is specified in the form ‘‘"*c*’’, where *c* represents the name of a buffer, one of the lowercase letters of the POSIX Locale. Specifying *buffer* shall cause the area of text affected by the command to be stored into the buffer as it was before the command took effect. This argument is also used on the ‘‘put’’ commands (`p` and `P`) to specify the buffer that shall provide the text to insert. When a command synopsis shows both [*buffer*] and [*count*] preceding the command letter, either can precede the other.

If the buffer name is specified in uppercase, and the buffer is to be modified (as with a deletion or yanking command) the buffer shall be appended to rather than being overwritten. If the buffer is not to be modified (as in a visual mode ‘‘put’’ commands) the buffer name can be specified in lowercase or uppercase with the same results. There shall be also one unnamed buffer, which is the repository for all text deleted (with the `delete` or visual mode `d` command) or yanked (with the `yank` or visual mode `y` command) when no buffer is specified.

There are also numbered buffers, 1 through 9, which shall be accessible only from visual mode. These buffers are special in that, in visual mode, when deleted text is placed in the unnamed buffer, it also shall be placed in buffer 1, the previous contents of buffer 1 shall be placed in buffer 2, and so on. Any text in buffer 9 shall be lost. Text that is yanked (or otherwise copied) into the unnamed buffer shall not modify the numbered buffers. Text cannot be placed directly into the numbered buffers although it can be retrieved from them by using a visual mode ‘‘put’’ command with the buffer name given as a number. When the *buffer* modifier is not used in the commands below, the unnamed buffer shall be the default.

column The (previous) value of the current column indicator.

count A positive integer used as an optional argument to most commands, either to give a size or a position (for display or movement commands), or as a repeat count (for commands that change text). This argument is optional and shall default to 1 unless otherwise noted in the individual command description.

end-of-line A description of the current column indicator value meaning that the current column indicator shall be set always to indicate the last character of the current line. Until the current column indicator is changed, the cursor shall be always positioned to the last character of whatever line is current.

line The (previous) value of the current line indicator.

nonblank A description of the current column indicator value meaning that the current column indicator shall be set to the position of the first nonblank character of the current line. If the current line has no nonblank characters, the indicator shall be set to the position of the last character of the line.

motion A command used as an optional trailing argument to some commands, indicating the extent of text to be affected by the command. The *motion* argument can be either the command character repeated or a cursor movement command. In the former case, exactly the current line shall be affected. In the latter case, the region specified shall be from the current cursor position to just before the cursor position indicated by the motion command. If the command operates on lines only, then all the lines that fall partly or wholly within this region shall be affected. Otherwise, the exact region as described above shall be affected. The following commands are considered to be cursor motion commands:

$	^	e	M
%	``	control-F	control-N
''	`*letter*	F	N
'*letter*	{	f	n
(	|	G	control-P
)	}	control-H	<space>
,	0	H	control-T
-	control-B	control-J	T
/	B	j	t
;	b	k	control-U
?	control-D	L	W
[[	control-E	l	w
]]	E	control-M	control-Y

The optional *count* prefix available for some of the motion com- 7
mands can be included and is considered part of the *motion* argu- 7
ment. For example, in c2w, the 2w is the *motion* argument. 7

previous context
A state set whenever a nonrelative motion command is executed in vi.

paragraph An area of text that begins with either the beginning of a file, an empty line, or section boundary and continues until either an empty line, section boundary, or the end of the editing buffer. Additional paragraph boundaries can be defined by the paragraph option. (See 5.10.7.5.12.)

section An area of text that starts with a line whose first character is either a `<form-feed>` or an open brace (`{`) and continues until the next section or the end of the editing buffer. The beginning and end of the editing buffer are also considered section boundaries. Additional section boundaries can be defined by the `sections` option. (See 5.10.7.5.18.)

sentence An area of text that begins with either the beginning of the editing buffer or the first nonblank character following the previous sentence, paragraph, or section boundary and continues until the end of the editing buffer or a period, exclamation point, or question mark (`.`, `!`, `?`) character followed by either an end of line or two `<space>`s. Any number of closing parentheses, brackets, or double-quote (`)`, `]`, `"`) characters can appear between the period, exclamation point, or question mark and the `<space>`s or end of line.

window The current value of the editor option `window`.

word In the POSIX Locale, `vi` recognizes two kinds of words: 7

— A maximal sequence of letters, digits and underscores, delim- 7
ited at both ends by: characters other than letters, digits, or 7
underscores; the beginning or end of a line; or the end of the 7
editing buffer. 7

— A maximal sequence of characters other than letters, digits, 7
underscores, or white space, delimited at both ends by: a 7
letter, digit, underscore, or white space; the beginning or end 7
of a line; or the end of the editing buffer. 7

! A character that can be appended to the command to modify its operation as detailed in the individual command descriptions.

After processing a command, if the current line and column indicators have been changed to specify a character not currently on the screen, the editor shall scroll or page the text to bring the needed character onto the screen. If the new current line is within *window* lines of the previous current line, and the terminal is capable of scrolling in the indicated direction, the text shall be scrolled the minimum number of lines necessary to present the needed character. Otherwise, the screen shall be redisplayed with the new current line positioned in the middle of the screen.

The following rules specify how exceptions shall be handled. When a movement command would cause a window to display beyond the beginning of the editing buffer, the window displayed is the window beginning with the first line of the editing buffer. When a movement command would cause a window to be displayed beyond the end of the editing buffer, the terminal shall be alerted and the command aborted. When the current line is empty, the cursor shall be placed in the first position of the line on the screen. When the current column indicator value is beyond the end of the current line, the cursor shall be placed on the last character of the current line, but the current column indicator shall be unchanged.

The following rules specify how position exceptions shall be handled. When the current column indicator is at the beginning (end) of a line and would be set to a position before the beginning (after the end) of the current line, the command shall not be executed, the terminal shall be alerted, and the current position shall remain unchanged. Otherwise, when the current column indicator would be set to a position before the beginning (after the end) of the current line, it shall be set instead to the position of the first (last) character of that line. When the current line indicator would be set to a position before the beginning (after the end) of the editing buffer, the command shall not be executed, the terminal shall be alerted, and the current position shall remain unchanged.

5.35.7.1.1 Page backwards

Synopsis : [*count*] `control-B`

Page backward in the file, allowing two lines of overlap, by displaying the window starting at line [*line* − (*count* ∗ (*window* −2))]. 7

Current line : The last line displayed.

Current column : Move to the first nonblank character of the current line or the first character if the line is a blank line.

5.35.7.1.2 Scroll forward

Synopsis : [*count*] `control-D`

Scroll forward *count* lines in the file. If *count* is not specified, the same number of lines shall be scrolled as by the previous `control-D` or `control-U` command. On the first `control-D` or `control-U` command, the amount scrolled shall be one-half the number of lines in a screenful. See 5.10.7.5.17.

Current line : (*line* + amount scrolled).

Current column : Move to the first nonblank character of the current line or the first character if the line is a blank line.

5.35.7.1.3 Scroll forward by line

Synopsis : [*count*] `control-E`

Scroll forward *count* lines, leaving the current line and column as is if possible.

Current line : Unchanged unless the current line scrolls off the screen; otherwise, move to the first line displayed.

Current column : Unchanged unless the current line scrolls off the screen; otherwise, move to the first character if it is a blank line or the last character of the line if the position of the current column is further into the line than the position of the last column of the line.

5.35.7.1.4 Page forward

Synopsis : [*count*] `control-F`

Page forward in the file, allowing two lines of overlap, by displaying the window starting at line [*line* + (*count* ∗ (*window* −2))]. 7

Current line : The first line displayed.

Current column : Move to the first nonblank character of the current line or the first character if the line is a blank line.

5.35.7.1.5 Display information

Synopsis : `control-G`

Display an informational message listing the current pathname, current line, number of lines, and other unspecified information, as does the `ex` command *file*.

Current line : Unchanged.

Current column : Unchanged.

5.35.7.1.6 Move cursor backwards

Synopsis : [*count*] `control-H`
Synopsis : [*count*] `h`

Move the cursor back *count* characters on the current line. No characters shall be erased from the screen by this command. In input mode, `control-H` shall have the exact same function.

Current line : Unchanged.

Current column : Set to (*column* − the number of columns occupied by *count* characters ending with the previous current column) or the beginning of the line if *count* is greater than the number of characters preceding the current character in the current line.

5.35.7.1.7 Move down in column

Synopsis : [*count*] `control-J`
Synopsis : [*count*] `j`
Synopsis : [*count*] `control-N`

Move the cursor down *count* lines without changing the current column.

Current line : *current line* + *count* . Unchanged if *count* is greater than the number of lines in the buffer following the current line.

Current column : Unchanged.

5.35.7.1.8 Clear and redisplay

Synopsis : `control-L`

Clear and redisplay the screen.

Current line : Unchanged.

Current column : Unchanged.

5.35.7.1.9 Move cursor down to nonblank

Synopsis : [*count*] `control-M`
Synopsis : [*count*] `+`

Move the cursor down *count* lines to the first nonblank character of that line.

Current line : *line* + *count* .

Current column : Move to the first nonblank character of the current line or the first character if the line is a blank line.

5.35.7.1.10 Move up in column

Synopsis : [*count*] `control-P`
Synopsis : [*count*] `k`

Move the cursor up *count* lines without changing the current column.

Current line : *line* − *count* .

Current column : Unchanged.

5.35.7.1.11 Redraw screen

Synopsis : `control-R`

Redraw the current screen. If any lines have been deleted from the logical screen and flagged as deleted on the terminal using the ‘‘@’’ convention (see 5.35.7), they shall be deleted, and the logical screen shall be redisplayed. Lines flagged with @ because they do not fit on the terminal display shall not be affected.

Current line : Unchanged.

Current column : Unchanged.

5.35.7.1.12 Move forward with tabs

Synopsis : `control-T`

Move the cursor forward *shiftwidth* (see 5.10.7.5.20) positions in insert mode by inserting `<tab>` characters, followed by space characters, as necessary, if the current cursor position is at the beginning of a line or preceded only by `<blank>`s.

Current line : Unchanged.

Current column: *column* + *shiftwidth*.

5.35.7.1.13 Scroll backward

Synopsis: [*count*] control-U

Scroll backward *count* lines in the file. If *count* is not specified, the same number of lines shall be scrolled as by the previous control-D or control-U command. On the first control-D or control-U command, the amount scrolled shall be the value of the scroll editor option; see 5.10.7.5.17.

Current line: (*line* − amount scrolled).

Current column: Move to the first nonblank character of the current line or the first character if the line is a blank line.

5.35.7.1.14 Escape next character

Synopsis: control-V *character*
Synopsis: control-Q *character* 8

Allow the entry of a subsequent *character*, removing any special meaning to the editor it has in input mode (e.g., <ESC>). The character ^ shall be displayed in the current cursor position until the subsequent character is typed, which then replaces the character ^.

Current line: Unchanged.

Current column: Unchanged.

5.35.7.1.15 Delete word

Synopsis: control-W

Delete the word preceding the cursor (including any <blank>s between the end of the word and the current cursor position) in input mode by moving the cursor back to the beginning of the word. No characters shall be erased from the screen by this command. Only text entered during the current input mode session and on the current line can be deleted with this command.

Current line: Unchanged.

Current column: Moved back as described above.

5.35.7.1.16 Scroll backward by line

Synopsis: [*count*] control-Y

Scroll backward *count* lines, leaving the current line and column as is, if possible.

Current line: Unchanged unless the current line scrolls off the screen; otherwise, the last line displayed.

Current column: Unchanged unless the current line scrolls off the screen; otherwise, move to the first character if it is a blank line or the last character of the line if the position of the current column is further into the line than the position of the last column of the line.

5.35.7.1.17 Terminate command or input mode

Synopsis : `<ESC>`

The effects of `<ESC>` depend on the mode and the command being entered, as fol- 8
lows: 8

— In command mode:

- If a line-oriented command (`/`, `?`, `:`, or `!`) is being entered, terminate the line-oriented command and execute it.

- If only part of a command has been entered, cancel the partial com-
mand. A command is not considered to be partially entered until at 7
least one non-*count* character has been entered. For example, 7
‘`33c<ESC>`’’ is partially entered and silently canceled, whereas 7
‘`33<ESC>`’’alerts the terminal (see the next item). 8

- Otherwise, alert the terminal.

— In input mode: Terminate input mode and return to command mode. At this point any text that was deleted in input mode, but that has not yet been erased from the screen, shall be erased from the screen; the current line shall be redisplayed to match exactly the changes made in input mode.

Current line : When a line-oriented command is being terminated, the current 8
line indicator shall be set as given for that command. Otherwise, unchanged.

Current column : When a line-oriented command is being terminated, the current 8
column indicator shall be set as given for that command; When terminating input 8
mode, if the cursor is not at the beginning of a line, the current column position
shall be set to (*column* − the width of the last character inserted). Otherwise, 8
unchanged. 8

8

5.35.7.1.18 Move cursor forward

Synopsis : [*count*] `<space>`
Synopsis : [*count*] `l` (ell)

Move the cursor forward *count* characters without changing the current line.

Current line : Unchanged.

Current column : (*current column* + the width of the next *count* characters) or the end of the line if *count* is greater than the number of characters in the line following the current column.

5.35.7.1.19 Replace text with results from shell command

Synopsis : [*count*] ! *motion shell-command(s)* `<newline>`

Pass the lines specified by *count* and *motion* as standard input to a shell command, for the program named in the **SHELL** environment variable, and replace those lines with the standard output of the shell command. An implementation may consider a *motion* command that does not result in an integral number of lines to be an error. After the *motion* is entered, the text of the command shall be prompted for on the last line of the display as described in 5.35.7. A warning shall be issued if the buffer has been changed since the last write.

Within the text of *command* , `%` and `#` shall be expanded as pathnames (the current and alternate pathnames, respectively), and `!` shall be replaced with the text of the previous `!` or `:!` command. (Thus, `!!` shall repeat the previous `!` command.) If no `!` or `:!` command has yet been executed, an informational message shall be displayed stating the problem.

The special meanings of `%`, `#`, and `!` can be overridden by escaping them with a backslash character. This command shall be affected by the editor options `autowrite` and `writeany`; see 5.10.7.5 for details.

Current line : The first line in *range* .

Current column : The first column of the replaced text.

5.35.7.1.20 Move cursor to end-of-line

Synopsis : [*count*] `$`

Move the cursor forward to the end of a line. The *count* argument specifies the number of lines, including the current line, to move forward.

Current line : *line* + *count* − 1.

Current column : *end-of-line* .

5.35.7.1.21 Move to matching character

Synopsis : `%`

Move the cursor to the parenthesis or curly brace matching the parenthesis or curly brace at the current position or on the current line forward from the current position. Matching shall be determined as follows: for a left parenthesis (respectively curly brace) the editing buffer shall be searched forward until either a left or right parenthesis is found. If a left parenthesis is found, a counter is incremented by one, and if a right parenthesis is found, a counter is decremented by one, and the search continues (until the end of the editing buffer is found). If a right parenthesis is found when the count is less than or equal to zero, it is the matching parenthesis. For a right parenthesis (respectively curly brace) the editing buffer shall be searched backward (to the beginning of the editing buffer), and each right parenthesis increments the counter, and each left parenthesis decrements it. When searching for parentheses, curly braces shall not be counted, and vice versa. If there is no matching character, the terminal shall be alerted and the current position shall remain unchanged.

Current line : Move to the line containing the matching character, as described above.

Current column : Move to the column containing the matching character, as described above.

5.35.7.1.22 Repeat substitution

Synopsis : &

Repeat the previous substitution command (equivalent to the ex & command; see 5.10.7.2.44) using the current line as the target. Flags specified on the previous substitution command shall be ignored by this command.

Current line : Unchanged.

Current column : Unchanged if the previous current column indicator value was *end-of-line* ; otherwise, move to the first nonblank character of the current line or the first character if the line is a blank line.

5.35.7.1.23 Return to previous context (at beginning of line)

Synopsis : ''
Synopsis : ' *letter*

Return to the previous context when followed by a ' character, if the context still exists, placing the cursor at the first nonblank character of the line or the last character if the line is a blank line. If the previous context no longer exists, the ' command shall have no effect. When followed by a lowercase letter from the POSIX Locale, return to the line marked with that letter (see the m command), at the first nonblank character in the line.

When used with an operator such as d to specify an extent of text, the operation takes place over complete lines. (See also 5.35.7.1.24.)

Current line : Move to the line from the previous context.

Current column : Move to the first nonblank character of the current line or the last character if the line is a blank line.

5.35.7.1.24 Return to previous context

Synopsis : ``
Synopsis : ` *letter*

When followed by a ` character, return to the previous context, placing the cursor at the character position marked. (The previous context shall be set whenever a nonrelative move is made.) When followed by a lowercase letter from the POSIX Locale, return to the line marked with that letter (see the m command), at the character position marked.

When used with an operator such as d to specify an extent of text, the operation takes place from the exact marked place to the current position within the line.

(See also 5.35.7.1.23.)

Current line : Move to the line from the previous context.

Current column : Set to the position of the character marked from the previous context.

5.35.7.1.25 Return to previous section

Synopsis : `[[`

Back up to the previous section boundary. This command shall be affected by the `sections` option; see 5.10.7.5.18. Note that the brackets in this command are part of the command syntax itself and are not metacharacters.

Current line : Move to the previous line that is a section boundary or to the first line of the editing buffer if no more section boundaries exist preceding the current line.

Current column : Move to the first nonblank character of the current line or the last character if the line is a blank line.

5.35.7.1.26 Move to next section

Synopsis : `]]`

Move forward to a section boundary. This command shall be affected by the `sections` option; see 5.10.7.5.18. Note that the brackets in this command are part of the command syntax itself and are not metacharacters.

Current line : Move to the next line that is a section boundary or to the last line of the editing buffer if no more section boundaries exist following the current line.

Current column : Move to the first nonblank character of the current line or the last character if the line is a blank line.

5.35.7.1.27 Move to first nonblank position on current line

Synopsis : `^`

Move to the first nonblank position on the current line.

Current line : Unchanged.

Current column : Move to the first nonblank character of the current line or the last character if the line is a blank line.

5.35.7.1.28 Move back to beginning of sentence

Synopsis : `[`*count*`] (`

Move backward to the beginning of a sentence. A *count* shall move back that many sentences.

Current line : Move to the line containing the beginning of the sentence.

Current column : Move to the first nonblank character of the sentence.

5.35.7.1.29 Move forward to beginning of sentence

Synopsis : [*count*] `)`

Move forward to the beginning of a sentence. A *count* shall move forward that many sentences.

Current line : Move to the line containing the beginning of the sentence.

Current column : Move to the first nonblank character of the sentence.

5.35.7.1.30 Move back to preceding paragraph

Synopsis : [*count*] `{`

Move back to the beginning of the preceding paragraph. A *count* specifies the number of paragraphs to move backward. This command shall be affected by the `paragraph` option; see 5.10.7.5.12.

Current line : Move to the line containing the beginning of the previous paragraph.

Current column : Move to the first nonblank character of the current line.

5.35.7.1.31 Move forward to next paragraph

Synopsis : [*count*] `}`

Move forward to the beginning of the next paragraph. A *count* specifies the number of paragraphs to move forward. This command shall be affected by the `paragraph` option; see 5.10.7.5.12.

Current line : Move to the line containing the beginning of the next paragraph.

Current column : Move to the first nonblank character of the current line.

5.35.7.1.32 Move to specific column position

Synopsis : [*count*] `|`

Place the cursor in the column position identified by *count* (if that position exists on the line). If *count* is omitted, it shall default to 1. If the length of the current line is less than *count* , the cursor shall be placed at the end of the current line. If the column position is spanned by a `<tab>` or a wide character, the cursor shall be placed on the character following the *count* -th column position.

Current line : Unchanged.

Current column : Move to the column position represented by *count* .

5.35.7.1.33 Reverse find character

Synopsis : [*count*] `,`

Reverse the last `f`, `F`, `t`, or `T` command, looking the other way in the current line. A *count* shall be equivalent to repeating the search that many times.

Current line : Unchanged.

Current column : Move to the first column position of the next character found.

5.35.7.1.34 Move to first nonblank of previous line

Synopsis : [*count*] `-`

Move to the first nonblank character in the previous line. A *count* specifies how many lines to move back.

Current line : *line* − *count*

Current column : Move to the first nonblank character of the current line or the last character if the line is a blank line.

5.35.7.1.35 Repeat

Synopsis : [*count*] `.`

Repeat the last command that changed the buffer. A *count* shall be passed on to the command being repeated.

Current line : Dependent on the results of the execution of the last command that changed the buffer.

Current column : Dependent on the results of the execution of the last command that changed the buffer.

5.35.7.1.36 Find regular expression

Synopsis : `/`

Prompt the user to enter a string on the last line on the screen, interpret it as a regular expression (see 5.10.7.3), and scan forward for the next occurrence of a matching string. The search shall begin when a `<newline>` or an `<ESC>` is entered to terminate the pattern; it can be prematurely terminated with an interrupt. 7

When used with an operator to specify an extent of text, the defined region is from the current cursor position to the beginning of the matched string. Whole lines can be specified by giving an offset from the matched line (using a closing `/` followed by a `+`*n* or −*n*).

Regular expressions are described in `ex`; see 5.10.7.3.

Current line : Move to the line in which the first (or next) regular expression match occurred.

Current column : Move to the column of the first character of the matched string.

5.35.7.1.37 Move to first character in line

Synopsis : `0` (zero)

Move to the first character on the current line. (The zero shall not be interpreted as a command when it is preceded by a nonzero digit.)

Current line : Unchanged.

Current column : Column position 1.

5.35.7.1.38 Execute an `ex` command

Synopsis : `:`

Begin an `ex` command. The `:` character, as well as the entered command, shall be displayed on the bottom line. The command shall be executed when the input
is terminated by entering a `<newline>` or an `<ESC>`. Typing a `control-V` (or 8
`control-Q`) character shall escape the following character, allowing its literal 8
value to be entered. 8

NOTE: This special property of `control-V` and `control-Q` is only effective in visual or open modes 8
and in `ex` commands called from visual/open mode with the `:` command.

If the `ex` command causes the screen to be scrolled or causes an escape to the shell, `vi` shall display a message indicating that it is waiting for a `<newline>` character. When a `<newline>` or an `<ESC>` character is entered, the screen shall be refreshed.

Current line : Dependent on the results of the execution of the `ex` command.

Current column : Dependent on the results of the execution of the `ex` command.

5.35.7.1.39 Repeat find

Synopsis : `[`*count*`] ;`

Repeat the last character find using `f`, `F`, `t`, or `T`. A *count* shall move the cursor to (or next to) the *count* -th occurrence of the character.

Current line : Unchanged.

Current column : Move to the column containing the character being searched for. If no matching character is found, it shall be unchanged.

5.35.7.1.40 Shift left

Synopsis : `[`*count*`] < [`*motion*`]`

Shift lines left one *shiftwidth* (see 5.10.7.5.20). The command can be followed by a motion command to specify lines. A *count* shall be passed through to the motion command.

When *motion* is <, it shall shift the current line (or *count* lines starting at the current one).

Current line : Unchanged.

Current column : Move to the first nonblank character of the line or the last character if the line is a blank line.

5.35.7.1.41 Shift right

Synopsis : **[** *count* **]** > **[** *motion* **]**

Shift lines right one *shiftwidth* (see 5.10.7.5.20). The command can be followed by a motion command to specify lines. A *count* shall be passed through to the motion command.

When *motion* is >, it shall shift the current line (or *count* lines starting at the current one).

Empty lines shall not be changed. 7

Current line : Unchanged.

Current column : Move to the first nonblank character of the line or the last character if the line is a blank line.

5.35.7.1.42 Scan backwards for regular expression

Synopsis : `?`

Scan backwards, the reverse of `/`. See 5.35.7.1.36.

Current line : Move to the line in which the next succeeding regular expression match occurred.

Current column : Move to the column of the first character of the matched string.

5.35.7.1.43 Execute 7

Synopsis : *@buffer* 7

Execute each line of the named buffer as one or more `vi` commands (including `ex` 7
commands, as described in 5.35.7.1.38). If the buffer name is @, then the last 7
buffer executed shall be used; if there is no last buffer, an error shall occur. The 7
text of a macro may contain an @ character calling another macro; however, recur- 7
sively calling the same macro produces undefined behavior. A `vi` error shall ter- 7
minate all currently executing macros. All changes made during a macro call 7
shall be treated as a unit and can be undone with a single `u` command. 7

Current line : Dependent on the results of the execution of the `vi` command(s). 7

Current column : Dependent on the results of the execution of the `vi` command(s). 7

5.35.7.1.44 Reverse case

Synopsis : [*count*] ~

Reverse the case of the *count* character(s) beginning at the current cursor position. Lowercase alphabetic characters shall be changed to uppercase and uppercase characters changed to lowercase. This command shall have no effect on nonalphabetic characters.

Current line : Unchanged.

Current column : Move to the next column unless it is on the last character of the line, in which case it shall remain unchanged.

5.35.7.1.45 Append

Synopsis : [*count*] `a`

Enter input mode, appending the entered text after the current cursor position. A
count shall cause the inserted text up to the first `<newline>` to be replicated that 8
many times; if a `<newline>` appears in the inserted text, only one occurrence of it 8
and any characters following it shall be inserted. For example, if the current line 8
is `abc`, the command 8

```
03afoo<newline>bar
```
8

changes that line to 8

```
afoofoofoo
barbc
```
8

Current line : Unchanged. 7

Current column : *column* + 1. 7

5.35.7.1.46 Append at end of line

Synopsis : [*count*] `A`

Append *count* copies of the input text at the end of the current line, equivalent to `$[`*count*] `a`. See the `a` command in 5.35.7.1.45.

Current line : Unchanged. 8

Current column : See 5.35.7.1.45.

5.35.7.1.47 Move backward to preceding word

Synopsis : [*count*] `b`

Move the cursor backward to the beginning of a word by repeating the following
algorithm *count* times: If the current position is at the beginning of a word, the 7
current position shall move to the first character of the preceding word. Otherwise, the current position shall move to the first character of the word at the current position. If no preceding word exists on the current line, the current position shall move to the first character of the last word on the first preceding line

that contains a word. For this command, an empty or blank line shall be considered to contain exactly one word.

Current line: Set to the line containing the word selected.

Current column: Set to the first character of the word selected.

5.35.7.1.48 Move backward to preceding bigword

Synopsis: [*count*] `B`

Move the cursor backward to the beginning of a bigword by repeating the following algorithm *count* times: If the current position is at the beginning of a bigword or the character at the current position cannot be part of a bigword, the current position shall move to the first character of the preceding bigword. Otherwise, the current position shall move to the first character of the bigword at the current position. If no preceding bigword exists on the current line, the current position shall move to the first character of the last bigword on the first preceding line that contains a bigword. For this command, an empty or blank line is considered to contain exactly one bigword.

Current line: Set to the line containing the bigword selected.

Current column: Set to the first character of the bigword selected.

5.35.7.1.49 Change

Synopsis: [*buffer*] [*count*] `c` *motion*

Delete the specified region of text and enter input mode to replace it with the entered text. If more than part of a single line is affected, the deleted text shall be saved in the numeric buffers. If only part of the current line is affected, then the last character to be deleted shall be marked with a $. A *count* shall be passed through to the motion command.

Current line: The current line and column position are dependent on the *motion* command following the `c`. For example, `cw` changes a word, `c[[` changes from the beginning of the current section, etc.

Current column: See *Current line*, above

5.35.7.1.50 Change to end-of-line

Synopsis: [*buffer*] [*count*] `C`

Change text from the current position through the end of line (*line* + *count* − 1); equivalent to [*buffer*] [*count*] `c$`.

Current line: See the `c` command in 5.35.7.1.49.

Current column: See 5.35.7.1.49.

5.35.7.1.51 Delete

Synopsis : [*buffer*] [*count*] `d` *motion* 8

Delete the specified region of text. If more than part of a line is affected, the text
shall be saved in the numeric buffers as well as any named [*buffer*]. A *count* 8
shall be passed through to the motion command.

Current line : The current line and column position are dependent on the *motion* command following the `d`. For example, `dw` deletes a word, `d$` deletes to the end of the line, etc.

Current column : See *Current line* , above.

5.35.7.1.52 Delete to end-of-line

Synopsis : [*buffer*] [*count*] `D` 8

Delete the text from the current position to the end of the current line; equivalent
to [*buffer*] [*count*] `d$`. 8

Current line : Unchanged.

Current column : Set to maximum of *column* − 1 or 1. 7

5.35.7.1.53 Move to end of word

Synopsis : [*count*] `e`

Move the cursor forward to the end of a word, by repeating the following algo- 7
rithm *count* times: If the current position is the end of a word, the current posi- 7
tion shall move to the last character of the following word. Otherwise, the current 7
position shall move to the last character of the word at the current position. If no 7
succeeding word exists on the current line, the current position shall move to the 7
last character of the first word on the next following line that contains a word. 7
For this command, an empty or blank line is considered to contain exactly one 7
word. 7

Current line : Set to the line containing the word selected.

Current column : Set to the last character of the word selected.

5.35.7.1.54 Move to end of bigword

Synopsis : [*count*] `E`

Move the cursor forward to the end of a bigword, by repeating the following algorithm *count* times: If the current position is the end of a bigword or the character at that position cannot be part of a bigword, the current position shall move to the last character of the following bigword. Otherwise, the current position shall move to the last character of the bigword at the current position. If no succeeding bigword exists on the current line, the current position shall move to the last character of the first bigword on the next following line that contains a bigword.

For this command, an empty or blank line is considered to contain exactly one big- 7
word. 7

Current line: Set to the line containing the bigword selected. 7

Current column: Set to the last character of the bigword selected. 7

5.35.7.1.55 Find character in current line (forward)

Synopsis: [*count*] `f` *character*

Scan the rest of the current line for the single character *character* and move the cursor to it if it is found. A *count* shall repeat the find that many times.

Current line: Unchanged.

Current column: Set to the column position containing the character that was scanned for. Unchanged if insufficient characters were found to satisfy the *count*.

5.35.7.1.56 Find character in current line (reverse)

Synopsis: [*count*] `F` *character*

Scan backwards in the current line for the single character *character*, moving the cursor to it if it is found. A *count* shall be equivalent to repeating the search that many times. If the search is unsuccessful, the terminal shall be alerted and the current column position shall remain unchanged.

Current line: Unchanged.

Current column: Set to the last character found moving backwards.

5.35.7.1.57 Move to line

Synopsis: [*count*] `G`

Move the cursor to line *count*, or to the last line of the editing buffer if *count* is not specified.

Current line: *count*, if specified; otherwise, the last line.

Current column: Move to the first nonblank character of the current line or the last character if the line is a blank line.

5.35.7.1.58 Move to top of screen

Synopsis: [*count*] `H`

Move the cursor to the line (*count* − 1) lines from the top of the current window (i.e., the *count*-th line, one-based, currently displayed).

Current line: *count* on the screen, as described above.

Current column: Move to the first nonblank character of the current line or the last character if the line is a blank line.

5.35.7.1.59 Insert before cursor

Synopsis : [*count*] `i` 8

Enter input mode, inserting the entered text before the cursor. (See also the `a`
command in 5.35.7.1.45.) A *count* shall cause the inserted text up to the first 8
`<newline>` to be replicated that many times; if a `<newline>` appears in the 8
inserted text, only one occurrence of it and any characters following it shall be 8
inserted. 8

Current line : Unchanged.

Current column : Unchanged.

5.35.7.1.60 Insert at beginning of line

Synopsis : [*count*] `I` 8

Enter input mode at the beginning of a line. This command shall behave identi-
cally to the `^`[*count*]`i` command. See 5.35.7.1.59. 8

Current line : Unchanged.

Current column : Move to the first nonblank character of the current line or the last character if the line is a blank line.

5.35.7.1.61 Join

Synopsis : [*count*] `J`

Join the current line with the next one, supplying appropriate spacing (in the 7
POSIX Locale): one `<space>` between words, two `<space>`s after a period, and no 7
`<space>`s at all when the last character of the first line is a `<blank>` or when the first character of the next line is `)`. In the case of lines ending with `<blank>`s, the `<blanks>` shall be retained, no spaces shall be added, and the lines shall be joined together. A *count* shall cause that many lines (minimum 2) to be joined, rather than 2. A count of 1 shall be treated as 2.

Current line : Unchanged.

Current column : Move to the character after the last character of the original line.

5.35.7.1.62 Move to bottom of screen

Synopsis : [*count*] `L`

Move the cursor to the first nonblank character of the last line on the screen. A *count* shall move to that line counting from the bottom. When used with an operator, whole lines shall be affected.

Current line : *last line* − *count*

Current column : Move to the first nonblank character of the current line or the last character if the line is a blank line.

5.35.7.1.63 Mark position

Synopsis : `m` *letter*

Mark the current position of the cursor with a single POSIX Locale lowercase letter *letter*. The exact position is referred to by `*letter*; the line is referred to by '*letter*.

Current line : Unchanged.

Current column : Unchanged.

5.35.7.1.64 Move to middle of screen

Synopsis : `M`

Move the cursor to the middle line on the screen, at the first nonblank position on the line.

Current line : Move to a line approximately in the middle of the display window.

Current column : Move to the first nonblank character of the current line or the last character if the line is a blank line.

5.35.7.1.65 Repeat regular-expression find (forward)

Synopsis : `n`

Repeat the last `/` or `?` scanning command.

Current line : Set to the line containing the character string found by the scan. Unchanged if no match was found.

Current column : Move to the first character of the character string found by the scan. Unchanged if no match was found.

5.35.7.1.66 Repeat regular-expression find (reverse)

Synopsis : `N`

Scan for the next match of the last pattern given to `/` or `?`, but in the reverse direction; this is the reverse of `n`.

Current line : Move to the matched line if a match is found or unchanged if no match is found.

Current column : Move to the first column position of the matched string or unchanged if no match is found.

5.35.7.1.67 Insert empty line below

Synopsis : `o`

Open a line below the current line and enter input mode.

Current line : The newly-created line, (*line* +1).

Current column: If the editor option `autoindent` is not set, move to the first column; otherwise, as in the description of `autoindent` in `ex` (see 5.10.7.5.1).

5.35.7.1.68 Insert empty line above

Synopsis: `O`

Open a new line above the current line and enter input mode.

Current line: Unchanged.

Current column: If the editor option `autoindent` is not set, move to the first column; otherwise, as described in the description of `autoindent` in `ex`.

5.35.7.1.69 Put from buffer following

Synopsis: [*buffer*] `p` 7

Insert text from buffer *buffer* following the current column or current line, as 7 described for the `P` command (see 5.35.7.1.70). If *buffer* is omitted, the unnamed buffer shall be used.

Current line: (*Current line* +1) if the buffer contains whole lines; otherwise, unchanged.

Current column: First nonblank character of the inserted text if the buffer contained whole lines; otherwise, the last character of the inserted text.

5.35.7.1.70 Put from buffer before

Synopsis: [*buffer*] `P`

Put the last deleted text back before/above the cursor. The text shall go back as whole lines above the cursor if it was deleted as whole lines. Otherwise, the text shall be inserted just before the cursor.

The command can be preceded by a named buffer specification (`"`*x*), to retrieve the contents of the buffer.

Current line: Unchanged.

Current column: Move to the last column position of the inserted characters. If the buffer contains whole lines, the current column shall be the first nonblank character of the inserted characters.

5.35.7.1.71 Enter `ex` mode

Synopsis: `Q`

Quit from `vi` and enter `ex` command mode.

Current line: Unchanged.

Current column: Unchanged.

5.35.7.1.72 Replace character

Synopsis : [*count*] r *character*

Replace a character on the screen with the character entered. If *count* is specified, the single character entered shall replace the current character and each of the next *count* − 1 characters in the line. (For example, 7rx changes the next seven characters to be xxxxxxx). Unlike the R command (see 5.35.7.1.73), entering a <newline> shall replace the character with a <newline>; however, when used with *count* , the next *count* characters in the line shall be replaced by a single <newline>.

Current line : Unchanged unless the replacement character is a <newline>, in which case the current line shall be incremented by 1.

Current column : Set to the last column position occupied by the replacement character, unless the replacement character is a <newline>, in which case the current column position shall be 1.

5.35.7.1.73 Replace characters

Synopsis : R

Replace characters on the screen with characters entered. If the end of the existing line is encountered, it shall be as if insert mode was entered at that point. If a <newline> is entered before the end of the existing line, it shall be entered into the editing buffer as if it was inserted, and replacement shall continue on the newly created line. No other line but the one in which the R command was given shall be affected.

Current line : Set to the line at the end of the replaced text.

Current column : Set to the end of the replaced text, if text was replaced. If no 7
text was replaced, set to the column position of the character after which text 7
would have been replaced if that character is on the current line; otherwise, 1. 7

5.35.7.1.74 Substitute character

Synopsis : [*buffer*] [*count*] s 8

Substitute *count* characters in the current line starting with the current column
position; equivalent to [*buffer*] [*count*] c<space>. 8

Current line : Unchanged.

Current column : Unchanged.

5.35.7.1.75 Substitute lines

Synopsis : [*buffer*] [*count*] S 8

Change whole lines (same as cc). A *count* shall change that many lines; *count* lines shall be deleted and vi shall enter input mode.

Current line : Unchanged.

Current column : Set to column position 1 or to the position indicated by the previous line if `autoindent` is set.

5.35.7.1.76 Move cursor to before character (forward)

Synopsis : [*count*] `t` *character*

Move the cursor forward within the current line to the character position just before a subsequent occurrence of *character* . A *count* shall place the cursor just before the *count* -th occurrence of the character when searching the line forwards. If the search is unsuccessful, the terminal shall be alerted and the cursor position shall remain unchanged.

Current line : Unchanged.

Current column : Move to the position before an occurrence of *character* following *column* (noninclusive).

5.35.7.1.77 Move cursor to after character (reverse)

Synopsis : [*count*] `T` *character*

Scan backwards in the current line for the single character *character* and if found, place the cursor just after that character. A *count* shall place the cursor just after the *count* -th occurrence of the character when searching the line backwards. If *count* -th occurrences of the character do not exist, the terminal shall be alerted and the current column position shall remain unchanged.

Current line : Unchanged.

Current column : Set to the column after the scanned character. Unchanged if the character was not found.

5.35.7.1.78 Undo

Synopsis : `u`

Reverse the last change made to the current buffer. If repeated, the command shall alternate between these two states, thus is its own inverse. When used after an insert that inserted text on more than one line, the lines shall be saved in the numeric named buffers.

Current line : Move to the position of the first line changed, if the reversal affects only one line or represents an addition or change; otherwise, move to the line preceding the deleted text.

Current column : Move to the first nonblank character on the updated current line.

5.35.7.1.79 Undo current line

Synopsis : `U`

Restore the current line to its state before the cursor was last moved to it.

Current line : Unchanged.

Current column : Set to column position 1 or to the position indicated by the previous line if `autoindent` is set.

5.35.7.1.80 Move to beginning of word

Synopsis : [*count*] `w`

Move the cursor forward to the beginning of a word by repeating the following
algorithm *count* times: If the current position is at the beginning of a word, the 7
current position shall move to the first character of the next word. If no subsequent word exists on the current line, the current position shall move to the first character of the first word on the first following line that contains a word.

For this command, an empty or blank line is considered to contain exactly one 7
word. 7

Current line : Set to the line containing the word selected. 7

Current column : Set to the first character of the word selected. 7

5.35.7.1.81 Move to beginning of bigword

Synopsis : [*count*] `W`

Move the cursor forward to the beginning of a bigword by repeating the following algorithm *count* times: If the current position is within a bigword or the character at that position cannot be part of a bigword, the current position shall move to the first character of the next bigword. If no subsequent bigword exists on the current line, the current position shall move to the first character of the first big-
word on the first following line that contains a bigword. For this command, an 7
empty or blank line is considered to contain exactly one bigword. 7

Current line : Set to the line containing the bigword selected.

Current column : Set to the first character of the bigword selected.

5.35.7.1.82 Delete character at cursor

Synopsis : [*buffer*] [*count*] `x` 8

Delete *count* characters in the current line starting with the current column posi-
tion; equivalent to [*buffer*] [*count*] `dl`. If the number of characters to be deleted 8
is greater than or equal to the number of characters to the end of the line, delete all of the characters from the current position to the end of the line and move the cursor to the new last character on the line.

Current line : Unchanged.

Current column : Set to the greater of (*current column* − the width of the *count* deleted characters) or 1.

5.35.7.1.83 Delete character before cursor

Synopsis : **[** *buffer* **] [** *count* **]** `X`

Delete the character before the cursor, equivalent to **[** *buffer* **] [** *count* **]** `dh`. A *count* shall repeat the effect, but only characters on the current line shall be deleted.

Current line : Unchanged.

Current column : Set to the greater of (*current column* − the width of the *count* deleted characters) or 1.

5.35.7.1.84 Yank

Synopsis : **[** *buffer* **] [** *count* **]** `y` *motion*

Copy (''yank'') the area of text specified by *count* and *motion* into the unnamed buffer, as well as into any named buffer specified by *buffer* .

Current line : Unchanged.

Current column : Unchanged.

5.35.7.1.85 Yank current line

Synopsis : **[** *buffer* **] [** *count* **]** `Y`

Place (yank) a *count* copies of the current line, equivalent to a **[** *buffer* **] [** *count* **]** `yy` command; see 5.35.7.1.84.

Current line : Unchanged.

Current column : Unchanged.

5.35.7.1.86 Redraw window

Synopsis : **[** *count1* **]** `z` **[** *count2* **]** *character*

Redraw the screen with a window *count2* lines long containing line *count1* placed as specified by *character* : `<newline>` specifies the top of the screen, `.` specifies the center of the screen, and `-` specifies the bottom of the screen. If *count1* is not specified, it shall default to the current line; if *count2* is not specified, it shall default to *window* .

Current line : *count1* , if specified; otherwise, unchanged.

Current column : Move to the first nonblank character of the current line or the first character if the line is a blank line.

5.35.7.1.87 Exit

Synopsis : `ZZ`

Exits the editor, writing out the buffer if it was changed since the last write to any file. This command shall behave identically to the `ex` command `xit`; see 5.10.7.2.38.

5.35.8 Exit Status

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

- 0 Successful completion.
- >0 An error occurred.

5.35.9 Consequences of Errors

Default.

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

Usage, Examples

The following table is a condensed version of information contained in the normative text. It is presented here to facilitate the review for accuracy of the `vi` commands and the editor options that affect those commands.

`vi` Command	Editor Option	Description
`control-B`	`window`	Determines screen size for paging.
`control-D`	`scroll`	Determines number of lines to be scrolled.
`control-F`	`window`	Determines screen size for paging.
`control-T`	`shiftwidth`	Move cursor forward shiftwidth positions.
`control-U`	`scroll`	Determines number of lines to be scrolled.
`{ }`	`paragraphs`	Defines additional paragraph boundaries for the `{` and `}` commands.
`[[ ]]`	`sections`	Defines additional section boundaries for the `[[` and `]]` commands.
`/`	`ignorecase`	Characters that have upper- and lowercase representations shall have those representations considered as equivalent for purposes of regular expression comparison.
	`magic`	When set, the characters `^ $ . * [ ~` have special meaning in regular expressions. When unset, only characters `^ $` have special

		meaning in regular expressions.
`<`	`shiftwidth`	Determines the number of character positions to be shifted.
`>`	`shiftwidth`	Determines the number of character positions to be shifted.
`?`	`ignorecase`	Characters that have upper- and lowercase representations shall have those representations considered as equivalent for purposes of regular expression comparison.
	`magic`	When set, the characters `^ $ . * [ ~` have special meaning in regular expressions. When unset, only characters `^ $` have special meaning in regular expressions.
`O, o`	`autoindent`	Starting indentation determined by the line on which the command is given.
`S`	`autoindent`	Starting indentation determined by the line being changed.
`ZZ`	`readonly`	Shall not allow a write to a file.
`z`	`window`	Redraw window with the number of lines equal to the default set by window.

The `control-L` and `control-R` commands differ in their treatment of deleted ‘‘@’’lines. The former always redisplays the screen and does not affect the status of any @ line; the latter redisplays only when needed to process the deleted @ lines.

When executing an individual `ex` command by entering `:`, it is not possible to enter a `<newline>` as part of the command because it is considered the end of the command. As explained in 5.10.7.4, a `control-M` preceded by a `control-V` (or `control-Q`) escape character in a regular expression substitution is mapped into a `<newline>`. A different approach is to enter `ex` command mode by using the `vi` `Q` command (and later resuming visual mode with the `ex` ‘`vi`’’command). In `ex` command mode, the single-line limitation does not exist. So, for example, the following is valid:

```
Q
s/break here/break\
here/
vi
```

The uses described for `control-V` can also be accomplished with `control-Q`, which is useful on terminals that use `control-V` for the down-arrow function. However, most historical implementations use `control-Q` for the *termios* START character, so the editor will generally not receive the `control-Q` unless `stty –ixon` mode is set. Any of the command characters described in this standard can be made ineffective by their selection as *termios* control characters, using the `stty` utility or other methods described in POSIX.1 {8}.

History of Decisions Made

Historically the control-B command skips two lines between each pair of pages it displays from a file. The working group viewed this as a bug and decided to go along with how the command was documented (as overlapping, not skipping, two lines) instead of standardizing the historical implementation.

See the Rationale for the ex utility for more information on vi.

A ‘‘restricted editor’’ (both the historical red utility and modifications to vi) were considered and rejected for inclusion. Neither option provided the level of security that many users might expect.

The [[,]], (,), }, and } commands are all affected by ‘‘section boundaries,’’ but in current implementations not all of the commands recognize the same section boundaries. This is considered to be a bug rather than a feature and a unique section-boundary algorithm was not described for each command.

The e and E commands described here differ slightly from some historical imple- 7
mentations in that an empty or blank line is described as containing exactly one 7
word or bigword, just as for b, B, w, and W. On some systems, E command skips 7
over empty/blank lines. The working group felt that a consistent definition of big- 7
word between the E command and the others was sufficient justification to change 7
one so that they all treat empty/blank lines the same way. 7

The = (reindent) command was omitted because it was LISP language-specific. (See the ex Rationale for the reasons LISP support was omitted.)

The text in Standard Output reflects the usage of the verb ‘‘display’’ in this clause; some vi implementations actually use standard output to write to the terminal, but POSIX.2 does not require that to be the case.

The action taken for signals other than SIGINT, SIGCONT, and SIGHUP is 7
unspecified because some implementations attempt to save the editing buffer in a 7
useful state when other signals are received. 7

5.36 `who` — Display who is on the system

5.36.1 Synopsis

```
who [−mTu]
```

5.36.2 Description

The `who` utility shall list various pieces of information about accessible users. The domain of accessibility is implementation defined.

The default output shall contain at least the following information for accessible users, in an unspecified format: the user's login name, terminal name, and time at which the user logged in.

5.36.3 Options

The `who` utility shall conform to the utility argument syntax guidelines described in 2.10.2.

The following options shall be supported by the implementation:

- `−m` Output only information about the current terminal.
- `−T` Show the state of each terminal, as described in 5.36.6.1.
- `−u` Output each displayed user's ''idle time'' in addition to any other 8
  information. The idle time is the time since any activity occurred on the user's terminal. The method of determining this is unspecified.

5.36.4 Operands

None.

5.36.5 External Influences

5.36.5.1 Standard Input

None.

5.36.5.2 Input Files

None.

5.36.5.3 Environment Variables

The following environment variables shall affect the execution of `who`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments).

LC_TIME This variable shall determine the format and contents of the date and time strings.

LC_MESSAGES This variable shall determine the language in which messages should be written.

5.36.5.4 Asynchronous Events

Default.

5.36.6 External Effects

5.36.6.1 Standard Output

The `who` utility writes its default information to the standard output in an implementation-defined format, subject only to the requirement of containing the information described above.

The following format shall be used for the `−T` option:

`"%s %c %s %s\n"`, *<user name>*, *<terminal state>*,
<terminal name>, *<time of login>*

where *<terminal state>* is one of the following characters:

`+` The terminal allows write access to other users.

`-` The terminal denies write access to other users.

`?` The terminal write-access state cannot be determined.

In the POSIX Locale, the *<time of login>* shall be equivalent in format to the output of:

```
date +"%b %e %H:%M"
```
7

If the −u option is used with −T, the idle time shall be added to the end of the previous format in an unspecified format.

5.36.6.2 Standard Error

Used only for diagnostic messages.

5.36.6.3 Output Files

None.

5.36.7 Extended Description

None.

5.36.8 Exit Status

The who utility shall exit with one of the following values:

0 Successful completion.

>0 An error occurred.

5.36.9 Consequences of Errors

Default.

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

Usage, Examples

None.

History of Decisions Made

Due to collisions with historical implementations, the options provided were a compromise to allow users to get at that functionality. The working group also weighed the idea of removing all the options, but felt these options offered users some real value.

It is recognized that the who command may be of limited usefulness, especially in a multilevel secure environment. It was felt, however, that having some standard method of determining the ''accessibility'' of other users would aid user portability.

The ''domain of accessibility'' was considered to be a sufficiently broad concept to permit interpretation either on a very secure basis or even to allow a network-wide 'rwho-like'' implementation.

The System V implementation of `who` is overloaded with some unrelated functionality (such as displaying the time of the last system reboot). The group decided that the prime function of `who` was to determine which users could be contacted on a system, by `write`, `talk`, etc., and therefore did not consider it proper to burden `who` with unrelated functionality.

The historical `who am I` command, while being one of the more intuitively obvious UNIX system commands, had to succumb to the tide of internationalization. It is replaced by the somewhat less charming `−m` option.

No format was specified for the default `who` output so that existing implementations should not need to be changed, except for adding the new options. In such an obviously user-oriented command, designed only for human consumption, this was not considered to be a deficiency.

The format of the terminal name is unspecified, but the descriptions of `ps`, `talk`, 7
and `write` require that they use the same format. 7

5.37 write — Write to another user

5.37.1 Synopsis

write *user_name* [*terminal*]

5.37.2 Description

The write utility shall read lines from the user's standard input and write them to the terminal of another user. When first invoked, it shall write the message:

> Message from
> *sender-login-id*
> (*sending-terminal*)
> [*date*]...

to *user_name* . When it has successfully completed the connection, the sender's terminal shall be alerted twice to indicate that what the sender is typing is being written to the recipient's terminal.

If the recipient wants to reply, this can be accomplished by typing

> write *sender-login-id* [*sending-terminal*]

upon receipt of the initial message. Whenever a line of input as delimited by a 8
NL, EOF, or EOL special character (see POSIX.1 {8} section 7.1.1.9) is accumulated 8
while in canonical input mode, the accumulated data shall be written on the other 8
user's terminal. Characters shall be processed as follows: 8

— Typing the <alert> character shall write the <alert> character to the recipient's terminal.

— Typing the erase and kill characters shall affect the sender's terminal in the manner described by the POSIX.1 {8} *termios* interface.

— Typing the interrupt or end-of-file characters shall cause write to write an appropriate message ("EOT\n" in the POSIX Locale) to the recipient's terminal and exit.

— Typing characters from LC_CTYPE classifications print or space shall 8
cause those characters to be sent to the recipient's terminal.

— When and only when the stty *iexten* local mode (see 4.59.4.4) is enabled, 8
additional special control characters and multibyte or single-byte functions 8
shall be processed in an implementation-defined manner. (See POSIX.1 {8} 8
section 7.1.1.9.) 8

— Typing other nonprintable characters shall cause implementation-defined sequences of printable characters to be written to the recipient's terminal.

To write to a user who is logged in more than once, the *terminal* argument can be used to indicate which terminal to write to; otherwise, the recipient's terminal is selected in an implementation-defined manner and an informational message shall be written to the sender's standard output, indicating which terminal was chosen.

Permission to be a recipient of a `write` message can be denied or granted by use
of the `mesg` utility (see 5.17). However, a user's privilege may further constrain 7
the domain of accessibility of other users' terminals. The `write` utility shall fail 7
when the user lacks the appropriate privileges to perform the requested action. 7

5.37.3 Options

None.

5.37.4 Operands

The following operands shall be supported by the implementation:

user_name Login name of the person to whom the message shall be written. This operand shall be of the form returned by the `who` utility (see 5.36).

terminal Terminal identification in the same format provided by the `who` utility.

5.37.5 External Influences

5.37.5.1 Standard Input

Lines to be copied to the recipient's terminal shall be read from standard input. 7

5.37.5.2 Input Files

None.

5.37.5.3 Environment Variables

The following environment variables shall affect the execution of `write`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).

LC_MESSAGES This variable shall determine the language in which diagnostic messages should be written.

5.37.5.4 Asynchronous Events

If an interrupt signal is received, `write` shall write an appropriate message on the recipient's terminal and exit with a status of zero. It shall take the standard action for all other signals; see 2.11.5.4.

5.37.6 External Effects

5.37.6.1 Standard Output

An informational message shall be written to standard output if a recipient is logged in more than once.

5.37.6.2 Standard Error

Used only for diagnostic messages.

5.37.6.3 Output Files

The recipient's terminal shall be used for output.

5.37.7 Extended Description

None.

5.37.8 Exit Status

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

0 Successful completion.

>0 The addressed user is not logged on, or the addressed user denies permission.

5.37.9 Consequences of Errors

Default.

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

Usage, Examples

None.

History of Decisions Made

The `write` utility was included in the standard since it can be implemented on all terminal types. The working group considered the `talk` utility, which cannot be implemented on certain terminals, to be a ''better'' communications interface. Both of these programs are in widespread use on historical implementations. Therefore, the working group decided that both utilities should be specified.

The format of the terminal name is unspecified, but the descriptions of `ps`, `talk`, 7
`who`, and `write` require that they all use or accept the same format. 7

Annex E
(informative)
Rationale and Notes

This annex summarizes the deliberations of the IEEE P1003.2 working group, the committee charged by the IEEE Computer Society's Technical Committee on Operating Systems and Application Environments with devising an interface standard for a shell and related utilities to support and extend POSIX.1 {8}.

The annex is being published along with the standard to assist in the process of review. It contains historical information concerning the contents of the standard and why features were included or discarded by the working group. It also contains notes of interest to application programmers on recommended programming practices, emphasizing the consequences of some aspects of the standard that may not be immediately apparent.

Just as this standard relies on the knowledge of architecture, history, and definitions from the POSIX.1 {8}, so does this annex. The reader is referred to the Rationale and Notes appendix of POSIX.1 {8} for background material and bibliographic information about UNIX systems in general and POSIX specifically, which will not be duplicated here.

Editor's Note: The text of most of the Rationale has been temporarily located in Sections 1 through 5, adjacent to the text it is explaining. The text will return to this annex after the completion of balloting.

E.4 Excluded Utilities

Many utilities were evaluated by the developers of the standard; more utilities were excluded from the standard than included. The following list contains some common UNIX system utilities that were considered by the developers, but not included as part of the UPE or POSIX.2.

Editor's Note: This list will eventually be integrated with Annex E.4 in the base standard.

`SCCS` Both SCCS and RCS were originally considered as part of a version control utilities portion of the scope. However, this aspect of the UPE was abandoned by the working group. Their was broad consensus that SCCS was the most widely implemented system and that it had some technical advantages over RCS (other than its difficult user interface). Some of the implementors objected to the notion of rewriting the very complicated SCCS utilities strictly from a standard description and indicated that they would oppose any such utilities in the standard unless their original author would put the source code into the public domain. The author was unwilling to do so. The IEEE may form a working group in the future to address this issue separately from the UPE.

`compress`, `uncompress`, `zcat`

These were proposed for an earlier draft, but met with too many objections to be included. The complex algorithm in historical implementations was expressed in the draft in terms of a C-language program, but many objections were raised concerning its accuracy and exact duplication of historical output (including one common historical ‘‘bug’’). Other objections stated that a C program was insufficient as a specification and that a detailed description in the Output Files subclause was needed instead. Consideration was given to removing just the algorithm and saying that `compress` was defined only at the user interface and was not necessarily portable between systems, but straw polls indicated that that would cause other objections as well. The working group found that it could not adequately address the requirements of the balloters on these points. In order to achieve consensus on an expeditious basis, these three utilities have been removed and could be considered for a future version of the standard; however, it is more likely that they would be part of the application portability sections, and not user portability.

`emacs` The community of `emacs` editing enthusiasts was quite adamant that the full `emacs` editor not be included in POSIX.2, because it was concerned that an attempt to standardize this very powerful environment would encourage vendors to ship versions conforming strictly to the standard, but lacking the extensibility required by the community. The author of the original `emacs` program also expressed his desire to omit the program. Furthermore, there were a number of historical UNIX systems that did not include `emacs`, or included it without supporting it, but there were very few that did not include and support `vi`. Since `emacs` is not in any of the four base documents, this finished any arguments concerning its inclusion.

`lint` This was proposed for an earlier draft, in the form of a `lint89` utility, but met with too many objections to be included. This technology is of declining importance because of the introduction of prototypes in the C Standard {7}. In particular, it is hard to specify the degree of checking for possible error conditions in programs in any compiler and specifying what `lint` would do in these cases is equally difficult.

It is fairly easy to specify what a compiler does. It requires specifying the language, what it does with that language, and stating that the interpretation of any incorrect program is unspecified. Unfortunately, with `lint`, it is required to specify what to do with erroneous programs. Since the number of possible errors and questionable programming practices is infinite, one cannot require `lint` to detect all errors of any given class. As the text of the drafts was tightened up, it was found that fewer and fewer useful statements could be made about `lint`.

Additionally, some vendors complained that since many compilers are distributed in a binary form without a `lint` facility (because the C Standard {7} does not require one), building the UPE from scratch will be much harder. Rather than being able to plug in a standard compiler component (simply by providing `c89` as an interface), source to that compiler would most likely need to be modified to provide the `lint` functionality. This was considered a major burden on system providers for a very small gain to developers (users).

`m4` The working group did not find that this macro processor had sufficiently wide usage for standardization. It is used in a small number of application installation tools and specialized applications, but it is rarely used by interactive users. Thus, it was omitted from this supplement as being out of the scope of work. There seems to be some interest within ISO to include `m4` in the final ISO version of POSIX.2, so it will be evaluated as part of the working group's P1003.2b revision effort.

`passwd` This utility was proposed for an earlier draft, but met with too many objections to be included. There were various reasons:

- Changing a password should not be viewed as a command, but as part of the login sequence. Changing one's password should only be done while a trusted path is in effect.
- Even though the text in previous drafts was intended to allow a variety of implementations to conform, the security policy for one site may differ from another site running with identical hardware and software. One site might use password authentication while the other did not. Vendors could not supply a `passwd` utility that would conform to the standard for all sites using their system.
- This is really a subject for the POSIX.7 system administration working group or POSIX.6 security working group.

`sdiff` This ''side-by-side diff'' utility has little portable application use and it was excluded from the UPE because it did not meet the requirement that it be part of one of the base documents.

`shar` Any of the numerous ‘‘shell archivers’’ were excluded from the UPE because they did not meet the requirement that they be parts of one of the base documents. Early drafts contained a `pshar` (‘POSIX shar’’) utility, but that was dropped in the face of the base document requirement (as well as considerable opposition to its complexity).

Annex F
(informative)

Balloting Instructions

This annex will not appear in the final standard. It is included in the draft to provide instructions for balloting that cannot be separated easily from the main document, as a cover letter might.

If you have received a copy of this draft before February 1992 it is important that you read this annex, whether you are an official member of the P1003.2a Balloting Group or not; comments on this draft are welcomed from all interested technical experts. **Your ballot is due to the IEEE office by 24 January 1992. This is not the date to postmark it—it is the date of receipt.**

Summary of Draft 8 Instructions

This is the third ''recirculation draft'' of P1003.2a. The recirculation procedure is 8
described in this annex. For this recirculation, we are accepting objections against any normative changes that occurred from Draft 7 to Draft 8 and the contents of the Unresolved Objections List, provided as a separate document from the draft.

Send your ballot and/or comments to:

IEEE Standards Office
Computer Society Secretariat
ATTN: P1003.2a Ballot (Anna Kaczmarek) 7
P.O. Box 1331
445 Hoes Lane
Piscataway, NJ 08855-1331

It would also be very helpful if you sent us your ballot in machine-readable form. Your official ballot must be returned via mail to the IEEE office; if we receive only the e-mail or diskette version, that version will not count as an official document. However, the online version would be a great help to ballot resolution. We can accept e-mail to the following address:

`hlj@Posix.COM` or `uunet!posix!hlj`

or IBM PC 3.5-inch/720K diskette (plain file) or Macintosh 3.5-inch diskette (plain text file [preferred], Word, or Write) or Sun-style QIC-24 cartridge tapes to:

Hal Jespersen, Chair P1003.2
POSIX Software Group
447 Lakeview Way
Redwood City, CA 94062

Some degree of judgment is required in determining what actually changed in Draft 8. Use the diff marks as a guide, but they will frequently mark text that has no real normative changes. Please limit your objections to the actual changes: for example, if we change the `foo –x` option to `–y`, don't use that as an opportunity to object that we have no `–z` option. Your objection should only address why the `x` to `y` change is a problem. (We have been balloting for a long time now and it is time to tighten the consensus and finish this up.) If you find problems unrelated to changes, submit them as comments and they will be considered seriously in that category. Thanks for your cooperation on this.

We are not accepting objections about how it is difficult to synchronize the descriptions between POSIX.2 Draft 11.2 and POSIX.2a Draft 8. We've chosen this slightly more difficult procedure to accelerate the balloting process, without waiting for the nits to be rung out of Draft 11.2. Rest assured that POSIX.2a will never be finally completed until after POSIX.2 is completely done.

Background on Balloting Procedures

The Balloting Group consists of over 160 technical experts who are members of the IEEE or the IEEE Computer Society; enrollment of individuals in this group
has already been closed. There are also a few ''parties of interest'' who are not 7
members of the IEEE or the Computer Society. Members of the Balloting Group 7
are required to return ballots within the balloting period. Other individuals who
may happen to read this draft are also encouraged to submit comments concern- 8
ing this draft. The only real difference between members of the Balloting Group and other individuals submitting ballots is that *affirmative* ballots are only
counted from Balloting Group members who are also IEEE or Computer Society 7
members. (There are minimum requirements for the percentages of ballots returned and for affirmative ballots out of that group.) However, objections and nonbinding comments must be resolved if received from any individual, as follows:

(1) Some objections or comments will result in changes to the standard. This will occur either by the publication of a list of changes or by the republication of an entire draft. The objections/comments are reviewed by a team from the P1003.2 working group, consisting of the Chair, Vice Chair, the Chair of the TCOS Standards Subcommittee, and one or more Technical Reviewers. The Technical Reviewers each have subject matter expertise in a particular area and are responsible for objection resolution in one or more sections.

(2) Other objections/comments will not result in changes.

 (a) Some are misunderstandings or cover portions of the document (front matter, informative annexes, rationale, editorial matters, etc.) that are not subject to balloting.

(b) Others are so vaguely worded that it is impossible to determine
what changes would satisfy the objector. These are referred to as
Unresponsive . (The Technical Reviewers will make a reasonable
effort to contact the objector to resolve this and get a newly worded
objection.) Further examples of unresponsive submittals are those
not marked as either *Objection* or *Comment* ; those that do not iden-
tify the portion of the document that is being objected to (each objec-
tion must be separately labeled); those that object to material in a 7
recirculation that has not changed and do not cite an unresolved 7
objection; those that do not provide specific or general guidance on
what changes would be required to resolve the objection.

(c) Finally, others are valid technical points, but they would result in
decreasing the consensus of the Balloting Group. (This judgment is
made based on other ballots and on the experiences of the working
group through almost five years of work and fifteen drafts preceding
this one.) These are referred to as *Unresolved Objections* . Sum-
maries of unresolved objections and their reasons for rejection are
maintained throughout the balloting process, are circulated to
members of the Balloting Group for their consideration, and are
presented to the IEEE Standards Board when the final draft is
offered for approval. Unresolved objections are only circulated to 8
the balloting group when they are presented by members of the bal- 8
loting group or by parties of interest. Unsolicited correspondence 8
from outside these two groups may result in draft changes, but are 8
not recirculated to the balloting group members. 8

Please ensure that you correctly characterize your ballot by providing one of the following:

(1) Your IEEE member number

(2) Your IEEE Computer Society affiliate number

(3) If (1) or (2) don't apply, a statement that you are a ''Party of Interest''

Ballot Resolution

The general procedure for resolving ballots is:

(1) The balloting cuts off on 24 January 1992. This is a receipt date at the IEEE , not a postmark date. (Please do not telephone or FAX on 24 January 1992 and say that your specific comments will come later; late-arriving comments will not be considered as objections.) We will accept comments after that date, including direct e-mail to the working group officers or the Technical Reviewers, but they will be treated as comments only—not objections. And we don't guarantee a written response to these late submissions.

(2) The ballots are put online and distributed to the Technical Reviewers.

(3) If a ballot contains an objection, the balloter will be contacted individually by telephone, letter, or e-mail and the corrective action to be taken

will be described (or negotiated). The personal contact will most likely not occur if the objection is very simple and obvious to fix or the balloter cannot be reached after a few reasonable attempts. Repeated failed attempts to elicit a response from a balloter may result in an objection being considered unresponsive, based on the judgment of the working group chair. Once all objections in a ballot have been resolved, it becomes an affirmative ballot.

(4) If any objection cannot be resolved, the entire ballot remains negative.

(5) Once more than seventy-five percent of the ballots received (that had voted either affirmative or negative) have been turned affirmative, two lists are published to the entire balloting group: the detailed list of approved changes and the list of unresolved objections, along with our reasons for rejecting them. This is known as a *recirculation* . You have minimum of ten days (after an appropriate time to ensure the mail got through) to review these two lists and take one of the following actions:

   (a) Do nothing; your ballots will continue to be counted as we have classified them, based on items (3) and (4).

   (b) Explicitly change your negative ballot to affirmative by agreeing to remove all of your objections from the unresolved list.

   (c) Explicitly change your affirmative ballot to negative based on your disapproval of either of the two lists you reviewed. If an issue is not on one of the two lists, new objections about this are not allowed. Negative ballots that come in on recirculations cannot be cumulative. They shall repeat any objections that the balloter considers unresolved from the previous recirculation. Ballots that simply say ‘‘and all the unresolved objections from last time’’ will be declared unresponsive. Ballots that are silent will be presumed to fully replace the previous ballot, and all objections not mentioned on the most current ballot will be considered as successfully resolved.

(6) The list of changes will frequently be a new draft document with the changes integrated. This is not a requirement, however, and a small number of changes may prompt merely a change list approach to recirculation.

(7) A copy of all your objections and our resolutions will be mailed to you. You can receive the full package of all resolutions from all ballots by contacting the IEEE Standards Office (who will probably charge you for the copying involved). If you don't agree with one of our resolutions and haven't been contacted personally before you receive this list, please accept our apologies and submit a new ballot against the new draft during the recirculation period.

(8) If at the end of the recirculation period there remain greater than seventy-five percent affirmative ballots, and no new objections have been received, a new draft is prepared that incorporates all the changes. This draft and the unresolved objections list go to the IEEE Standards Board

for approval. If the changes cause too many ballots to slip back into negative status, another resolution and recirculation cycle begins.

Balloting Guidelines

This section consists of guidelines on how to write and submit the most effective ballot possible. The activity of resolving balloting comments is difficult and time consuming. Poorly constructed comments can make that even worse.

We have found several things that can be done to a ballot that make our job more difficult than it needs to be, and likely will result in a less than optimal response to ballots that do not follow the form below. Thus it is to your advantage, as well as ours, for you to follow these recommendations and requirements.

If a ballot that significantly violates the guidelines described in this section comes to us, we will determine that the ballot is unresponsive, and simply ignore all the material in it.

Secondly, objections that don't contain a specification so that the correction to resolve the objection ''can be readily determined'' are also unresponsive and will be ignored.

(If we do recognize a ballot that is generally ''unresponsive,'' we will try to inform the balloter as soon as possible so he/she can correct it, but it is ultimately the balloter's responsibility to assure the ballot is responsive.)

Typesetting is not particularly useful to us. And please do not send handwritten ballots. Typewritten (or equivalent) is fine, and if some font information is lost it will be restored by the Technical Editor in any case. If you use `nroff`, you will include extraneous spacing and sometimes backspaces and overstrikes; if you really must use `nroff`, please turn off hyphenation and line adjusting:

```
.hy 0
.na
```

and run the output through `col -b` to remove all the overstrikes. (Also remember that backslashes and leading periods and apostrophes in your text will 7
be treated impolitely by the `*roff` family). The ideal ballot is formatted as a ''flat ASCII file,'' without any attempt at reproducing the typography of the draft and without embedded control characters or overstrikes; it is then printed in Courier (or some other typewriter-like) font for paper-mailing to the IEEE Standards Office and simultaneously e-mailed to the working group Chair.

Don't quote others' ballots. Cite them if you want to refer to another's ballot. If more than one person wants to endorse the same ballot, send just the cover sheets and one copy of the comments and objections. [Note to Institutional Representatives of groups like X/Open, OSF, UI, etc.: this applies to you, too. Please don't duplicate objection text with your members.] Multiple identical copies are easy to deal with, but just increase the paper volume. Multiple almost-identical ballots are a disaster, because we can't tell if they are identical or not, and are likely to miss the subtle differences. Responses of the forms:

— ‘‘I agree with the item in <someone>’s ballot, but I’d like to see this done instead’’

— ‘‘I am familiar with the changes to `foo` in <someone>’s ballot and I would object if this change is [or is not] included’’

are very useful information to us. If we resolve the objection with the original balloter (the one whose ballot you are referencing), we will also consider yours to be closed, unless you specifically include some text in your objection indicating that should not be done.

Be very careful of ‘‘Oh, by the way, this applies <here> too’’ items, particularly if they are in different sections of the document that are likely to be seen by different reviewers. They are probably going to be missed! Note the problem in the appropriate section, and cite the detailed description if it’s too much trouble to copy it. The reviewers don’t have time to read the whole ballot, and only read the parts that appear to apply to them. Particularly where definitions are involved, even if the change really belongs in one section but the relevant content is in another, an extra cross-reference would be indicated. If you wish to endorse someone else’s ballot, either in whole or part, be specific about whether you will be automatically satisfied if they are satisfied. If you will not necessarily be satisfied if they are, your ballot could be deemed unresponsive because it does not give achievable conditions under which your ballot could be converted to affirmative. You then must give the conditions under which you would be satisfied as well. If you would be satisfied in some areas and not in others, it is best to specifically point to each specific objection in the ballot you point to, giving the conditions for each.

Please consider this a new ballot that should stand on its own. Please do not make backward references to your ballots for previous drafts—include all the text you want considered here, because the Technical Reviewer may not have your old ballot. And, the old section and line numbers won’t match up anyway. If one of your objections was not accepted exactly as you wanted, it will not be useful to send in the exact text you sent before; read the nearby Rationale section and come up with a more compelling (or clearly-stated) justification for the change.

Please be very wary about global statements, such as ‘‘all of the arithmetic functions need to be defined more clearly.’’ Unless you are prepared to cite specific instances of where you want changes made, with reasonably precise replacement language, your ballot will be considered unresponsive.

Ballot Form

The following form is recommended. We would greatly appreciate it if you sent the ballot in electronic form in addition to the required paper copy. Our policy is to handle all ballots online, so if you don’t send it to us that way, we have to type it in manually. For the last POSIX.2 ballot, only one or two balloters could not accommodate us on this and thus we had very little typing to do. See the first page of this Annex for the addresses and media. As you’ll see from the following, formatting a ballot that’s sent to us online is much simpler than a paper-only ballot.

The ballot should be page-numbered, and contain the name, e-mail address, and phone number(s) of the objector(s). (If you send us only a paper copy, make sure this information appears on every page; electronic ballots just need it once, in the beginning.) The lines before the first dashed line are a page header, and should only appear once on each page. Please leave adequate (at least one inch) margins on both sides. Each objection/comment/editorial comment should be sequentially numbered, not in individual ranges [i.e., not Objection #1, Comment #1]

Since we deal with the ballots online, there is no longer any requirement to put only one objection or section per page.

Don't format the ballot as a letter or document with its *own* section numbers. These are simply confusing. As shown below, it is best if you cause each objection and comment to have a sequential number that we can refer to amongst ourselves and to you over the phone. Number sequentially from 1 and count objections, comments, and editorial comments the same; don't number each in its own range. If you don't do this, we'll number them ourselves, but you won't know what numbers we're using.

Please precede each objection/comment with a little code line (if you don't, we'll have to do it ourselves):

@ *<section>* . *<clause>* *<code>* *<seqno>*

where:

@	At-sign in column 1 (which means no @'sin any other column 1's).
<section>	The major section (chapter or annex) number or letter in column 3. Use zero for Global or for something, like the frontmatter, that has no section or annex number.
<clause>	The clause number (second-level header). Please do not go deeper than these two levels. In the text of your objection or comment, go as deep as you can in describing the location, but this code line uses two levels only.
<code>	One of the following lowercase letters, preceded and followed by spaces: o Objection. c Comment or Editorial Comment.
<seqno>	A sequence number, counting all objections and comments in a single range.

Objection:

```
Balloter Name             (202)555-1212       page  x of nn.
E-Mail Address            FAX:   Fax Number
Balloter2 Name                   (303)555-1213
E-Mail Address2           FAX:   Fax Number2
------------------------------------------------------------------
@ x.y o seq#
<Seq#> Sect x.y OBJECTION. page xxx, line zzz:
```

`Problem:`

A clear statement of the problem that is observed, sufficient for others to understand the nature of the problem. (Note that you should identify problems by section, page, and line numbers. This may seem redundant, but if you transpose a digit pair, we may get totally lost without a cross-check like this. Use the line number where the problem starts, not just where the section itself starts; we sometimes attempt to sort objections by line numbers to make editing more accurate. If you are referring to a range of lines, please don't say ‘‘lines 1000ff;’’ use a real range so we can tell where to stop looking. We will really love you if you can manage to include enough context information in the problem statement (such as the name of the utility) so we can understand it without having the draft in our laps at the time. (It also helps you when we e-mail it back to you.) If you are objecting to an action in the Unresolved Objections List, use the section/page/line number reference for the appropriate place in the standard; don't refer to the UOL 7
except to cite its number and for clarification of your points. 7

`Action:`

A precise statement of the actions to be taken on the document to resolve the objection above, which if taken verbatim will completely remove the objection.

If there is an acceptable range of actions, any of which will resolve the problem for you if taken exactly, please indicate all of them. If we accept any of these, your objection will be considered as resolved.

If the Action section is omitted or is vague in its solution, the objection will be reclassified as a nonbinding comment. The Technical Reviewers, being human, will give more attention to Actions that are well-described than ones that are vague or imprecise. The best ballots of all have very explicit directions to substitute, delete, or add text in a style consistent with the rest of the document, such as:

```
Delete the sentence on lines 101-102:

      "The implementation shall not ... or standard error."

On line 245, change "shall not" to "should not".

After line 103, add:

       -r    Reverse the order of bytes read from the file.
```

Some examples of poorly-constructed actions:

```
Remove all features of this command that are not supported by BSD.

Add -i.

Make this command more efficient and reliable.

Use some other flag that isn't so confusing.

I don't understand this section.

Specify a value--I don't care what.
```

Objection Example:

```
Hal Jespersen             (415) 364-3410            page  3 of 17.
UUCP: hlj@Posix.COM  FAX:   (415) 364-4498
-------------------------------------------------------------------
@ 2.6 o 23
23. Sect 2.6 OBJECTION. page 77, line 1217:

Problem:

The EDITOR environment variable is not used as stated
in my company.  This description would cause hundreds
of my shell scripts to break.

Action:

Change the first sentence on line 1217 to:

       The e-mail address of the editor of the user's
       favorite POSIX standard.

-----------------------
@ 3.1 o 24
24. Sect 3.1.6 OBJECTION. page 123, line 17:

Problem:

I support UO 3.01-999-6 concerning the objection to the              7
definition of "operator".                                            7
This definition would cause great hardship to the users              7
of the systems I develop.                                            7
I feel your rationale for rejection was inappropriate                7
because you overlooked the following technical points [etc.]...      7

Action:

Change the term "operator" to "operation-symbol" in this
definition and globally throughout Section 3.
```

Comment:

```
-------------------------------------------------------------------
@ x.z c seq#
<Seq#> Sect x.z COMMENT. page xxx, line zzz:
```

A statement of a problem that you might want to be resolved by the reviewer, but which does not in any way affect whether your ballot is negative or positive. The form for objections is not required, but it increases the probability that your comment will have an effect on the final document.

Although there may be questions to you or responses on the topic, no changes in the drafts are required by a comment, although it will be looked at to determine whether the concern should be addressed. It is possible to abuse this rule and label all of your comments as objections, but it is a significant disservice to the individuals who are volunteering their time to address your concerns.

Remember that any issue concerning the pages preceding page 1 (the Frontmatter), Rationale text with shaded margins, Annexes, NOTES in the text, footnotes, or examples will be treated as a nonbinding comment whether you label it that way or not, but it would help us if you'd label it correctly.

Editorial Comment:

```
------------------------------------------------------------
@ x.z c seq#
<Seq#> Sect x.z EDITORIAL COMMENT. page xxx, line zzz:
```

These are for strictly editorial issues, where the technical meaning of the document is not changed. Examples are: typos; misspellings; English syntax or usage errors; appearances of lists or tables; arrangement of sections, clauses, and subclauses (except where the location of information changes the optionality of a feature). Marking these as comments but indicating that they are editorial speeds the process.

Please be aware that after balloting concludes the document will be subjected to more sets of editors at the IEEE and ISO who are empowered to make broad editorial changes and rewording (for example, to get the text ready for translation into French.)

Thank you for your cooperation in this important balloting process.

Hal Jespersen

Identifier Index

Alphabetic Topical Index

C

D

E

F

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Acknowledgments

We wish to thank the following organizations for donating significant computer, printing, and editing resources to the production of this standard: Amdahl Corporation, AT&T, Concurrent Computer Corporation, the POSIX Software Group, UniSoft Corporation, and the X/Open Group.

This document was also approved by ISO/IEC JTC 1/SC22/WG15 as ISO/IEC 9945-2:199x. The IEEE wishes to thank the advisory groups of the National Bodies participating in WG15 for their contributions: Austria, Belgium, Canada, Denmark, France, Germany, Japan, Netherlands, United Kingdom, USA, and USSR.

The IEEE also wishes to thank the delegates to WG15 for their contributions:

AUSTRIA
Gerhard Schmitt
Wolfgang Schwabl

CANADA
Joe Cote
Patrick Dempster
George Kriger
Bernard Martineau
Major Douglas J. Moore
Arnie Powell
Paul Renaud
Richard Sniderman

CEC
Phil Bertrand
Manuel Carbajo
Michel Colin

DENMARK
Peter E. Cordsen
Isak Korn
Keld Simonsen
Claus Tondering

FINLAND
Jikka Haikala

FRANCE
Pascal Beyles
Christophe Binot
Claude Bourstin
Jean-Michel Cornu
Yves Delarue
Eric Dumas
Maurice Fathi
Gerald Krummeck
Herve Schauer
Hubert Zimmerman

GERMANY
Ron Elliot
Helmut Stiegler
Claus Unger
Rainer Zimmer

IRELAND
Hans-Jurgen Kugler

JAPAN
Hiromichi Kogure
Shigekatsu Nakao
Yasushi Nakahara
Nobuo Saito

NETHERLANDS
J. Van Katwijk
Willem Wakker
H.J. Weegenaar

SWEDEN
Mat Linder

UK
Nigel Bevan
Cornelia Boldyreff
Dave Cannon
Don Chacon
Dominic Dunlop
David Flint
Don E. Folland
Martin Kirk
Neil Martin
Brian Meek
Kevin Murphy
Ian Newman
Philip Rushton

USA
Robert Bismuth
Steven L. Carter
Terence S. Dowling
Ron Elliott
Dale Harris
John Hill
James D. Isaak
Hal Jespersen
Roger J. Martin
Shane McCarron
Barry Needham
Donn S. Terry
Alan Weaver

USSR
V. Koukhar
Ostapenko Georgy Pavlovich

Also we wish to thank the organizations employing the members of the Working Group and the Balloting Group for both covering the expenses related to attending and participating in meetings, and donating the time required both in and out of meetings for this effort.

Editor's Note: This list is the union of the companies sponsoring Working Group attendees and Balloting Group members. Any corrections should be sent to the Technical Editor, bypassing the normal balloting process.

3M Company
ACE Associated Computer Experts b.v.
Aeon Technologies, Inc.
Alis Development
Amdahl Corporation *
Apollo Computer Inc.
Apple Computer Inc.
Ardent Computer
AT&T *
AT&T Bell Laboratories
AT&T UNIX Pacific Co., Ltd.
Axon Data Information Systems
BBN Communications Corporation
Bell Communications Research
BNR, Inc.
Bolt Beranek & Newman
BP Research International
British Telecom
British Telecom Research Labs
Charles River Data Systems *
Chemical Abstracts Service
Chorus Systemes
Commission of the European Communities
Computer X, Inc.
Concurrent Computer Corporation *
Control Data Corporation *
Convergent Technologies
Convex Computer Corporation
Cray Research, Inc. *
Cyber-Dyne, Inc.
Datapoint Corporation
Data General Corporation
Digital Equipment Corporation *
Digital ETU
Douglas Aircraft Company
Electrospace Systems, Inc.
Emerging Technology Group Inc.
Emory University Computing Center
Encore Computer
ETA Systems, Inc.
Federal Judicial Center
Ford Motor Company
Free Software Foundation
General Electric Corporation
Georgia Institute of Technology
Gilbert International Inc.
Gould CSD
Harris Corporation
HCR Corporation
Hewlett-Packard Company *
Honeywell Bull, Inc.
HQ USAISC
Hughes Aircraft Co.
IBM Corporation *
IBM Japan
IBM Systems Integration Division
Icon International, Inc.
Intel
Mallinckrodt Institute
Martin Marietta Data Systems
Masscomp
Mercury Computer Systems
Microsoft Corporation
Mindcraft, Inc.
Mitsubishi Electric Corporation
Mortice Kern Systems Inc.
Motorola Inc.
Myrias Research Corporation
NAPS International
NASA-KSC
National Institute of Standards and Technology
Naval Postgraduate School
NCR Corporation *
Northern Telecom, Inc.
Novell, Inc.
Ohio State University
Pacific Marine Tech
POSIX Software Group
PRC
Prime Computer, Inc.
R&D Associates
Rabbit Software Corporation
ROLM Mil Spec Computers
Sandia National Laboratory *
Santa Cruz Operation Inc.
Saudia National Labs
Sequent
Shia Systems Inc.
Simpact Associates, Inc.
SoHar
Sphinx Ltd.
St. Lawrence College
Stellar Computer Inc.
Sun Microsystems, Inc. *
Syntactics
Tandem Computers Inc.
Technical University of Vienna
Tektronix, Inc. *
Texas Instruments
The Instruction Set Ltd.
The MITRE Corp
Toshiba Corporation
UFP b-GRC
UniForum *
UniSoft Corporation
UniSoft Ltd.
Unisys Corporation *
University Of California, Berkeley
University of Hong Kong
University of Indonesia
University of Maryland
University of Texas at Arlington
University of Utah
University of Victoria
University of Vienna
USAF

Interactive Systems Corporation
Ironwood Software
KAIST
Lachman Associates, Inc.
Lawrence Livermore National Lab
Loral Rolm MilSpec Computers
Lotus Development Corporation
Mahavishi International University
Whitesmiths, Ltd.
Woods Hole Oceanographic Inst.
USAISEC
USENIX Association*
US Army
US Army Ballistic Research Lab.
US Army Computer Engineering Center
US West Advanced Technologies
Videoton
Wang Laboratories
X/Open Company Ltd.
XIOS Systems Corporation

In the preceding list, the organizations marked with an asterisk (*) have hosted 1003 Working Group meetings since the group's inception in 1985, providing useful logistical support for the ongoing work of the committees.