IEEE P1003.2a Draft 8 – December 1991

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All rights reserved as an unpublished work.

This is an unapproved and unpublished IEEE Standards Draft, subject to change.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

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

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

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

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

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

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

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

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.

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

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 7
`c89` produce unspecified results. 7

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 7
`c89` produce unspecified results. 7

file `.f` Files with basenames ending with the `.f` suffix shall be treated
as FORTRAN language source code. Such files that are not valid 7
input to `fort77` produce unspecified results. 7

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 deter- 8
mine the order in which the tags file is written. 8

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 7

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

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

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

5.7.7 Extended Description

If the operand identifies C-language source, the `ctags` utility shall attempt to produce an output line for each of the following objects:

— Function definitions

— Typedefs

— Macros with arguments

It may also produce output for any of the following objects:

— Function prototypes

— Structures

— Unions

— Global variable definitions

— Enums

— Macros without arguments

— `#define` statements

— `#line` statements

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 Utilities Option in Annex A, `ctags` produces unspecified results for C-language source code files. It should write to standard error a message identifying this condition and cause a nonzero exit status to be produced.

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 Annex C, `ctags` produces unspecified results for FORTRAN source code files. It should write to standard error a message identifying this condition and cause a nonzero exit status to be produced.

It is implementation defined what other objects (including duplicate identifiers) produce output.

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

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

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

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
```

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

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.

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

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*>

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

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

<space used>/(*<space used>*+*<space free>*)

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.

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

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 4.2 BSD `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

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

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.

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

`−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.

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

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.

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

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.

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

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

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

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.

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

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_**.

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

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 terminal 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).

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

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.

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

During initialization, before the first file is read or any user commands from the 8
terminal are processed, if the environment variable **EXINIT** is set, the editor shall 8
execute the `ex` commands contained in that variable. If the variable is not set, `ex` 8
shall attempt to read commands from the file `$HOME/.exrc` (the file `.exrc` in the 8
directory referred to by the **HOME** environment variable). If and only if **EXINIT** 8
or `$HOME/.exrc` sets the editor option `exrc`, `ex` finally shall attempt to read com- 8
mands from a file `.exrc` in the current directory. In the event that **EXINIT** is not 8
set and the current directory is the user's home directory, any `.exrc` file shall 8
only be processed once. No `.exrc` file shall be read unless it is owned by the 8
same user ID as the effective user ID of the process. After any `.exrc` files are pro- 8
cessed, any commands specified by the `−c` option shall be processed. 8

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 dis-
cussed under the description of each command. When the buffer contains no 7
lines, the current line shall be set to zero. 7

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

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