Chapter 1

Introduction

1.1 Scope

The scope of IEEE Std 1003.1-200x is described in the Base Definitions volume of IEEE Std 1003.1-200x.

1.2 Conformance

Conformance requirements for IEEE Std 1003.1-200x are defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 2, Conformance.

1.3 Normative References

Normative references for IEEE Std 1003.1-200x are defined in the Base Definitions volume of IEEE Std 1003.1-200x.

1.4 Change History

Change history is described in the Rationale (Informative) volume of IEEE Std 1003.1-200x, and in the CHANGE HISTORY section of reference pages.

1.5 Terminology

This section appears in the Base Definitions volume of IEEE Std 1003.1-200x, but is repeated here for convenience:

For the purposes of IEEE Std 1003.1-200x, the following terminology definitions apply:

can

Describes a permissible optional feature or behavior available to the user or application. The feature or behavior is mandatory for an implementation that conforms to IEEE Std 1003.1-200x. An application can rely on the existence of the feature or behavior.

implementation-defined

Describes a value or behavior that is not defined by IEEE Std 1003.1-200x but is selected by an implementor. The value or behavior may vary among implementations that conform to IEEE Std 1003.1-200x. An application should not rely on the existence of the value or behavior. An application that relies on such a value or behavior cannot be assured to be portable across conforming implementations.

The implementor shall document such a value or behavior so that it can be used correctly by an application.

legacy

Describes a feature or behavior that is being retained for compatibility with older applications, but which has limitations which make it inappropriate for developing portable

applications. New applications should use alternative means of obtaining equivalent functionality.

may

Describes a feature or behavior that is optional for an implementation that conforms to IEEE Std 1003.1-200x. An application should not rely on the existence of the feature or behavior. An application that relies on such a feature or behavior cannot be assured to be portable across conforming implementations.

To avoid ambiguity, the opposite of *may* is expressed as *need not*, instead of *may not*.

shall

For an implementation that conforms to IEEE Std 1003.1-200x, describes a feature or behavior that is mandatory. An application can rely on the existence of the feature or behavior.

For an application or user, describes a behavior that is mandatory.

should

For an implementation that conforms to IEEE Std 1003.1-200x, describes a feature or behavior that is recommended but not mandatory. An application should not rely on the existence of the feature or behavior. An application that relies on such a feature or behavior cannot be assured to be portable across conforming implementations.

For an application, describes a feature or behavior that is recommended programming practice for optimum portability.

undefined

Describes the nature of a value or behavior not defined by IEEE Std 1003.1-200x which results from use of an invalid program construct or invalid data input.

The value or behavior may vary among implementations that conform to IEEE Std 1003.1-200x. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.

unspecified

Describes the nature of a value or behavior not specified by IEEE Std 1003.1-200x which results from use of a valid program construct or valid data input.

The value or behavior may vary among implementations that conform to IEEE Std 1003.1-200x. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.

1.6 Definitions

Concepts and definitions are defined in the Base Definitions volume of IEEE Std 1003.1-200x.

1.7 Relationship to Other Documents

1.7.1 System Interfaces

This subsection describes some of the features provided by the System Interfaces volume of IEEE Std 1003.1-200x that are assumed to be globally available by all systems conforming to this volume of IEEE Std 1003.1-200x. This subsection does not attempt to detail all of the features defined in the System Interfaces volume of IEEE Std 1003.1-200x that are required by all of the utilities defined in this volume of IEEE Std 1003.1-200x; the utility and function descriptions point out additional functionality required to provide the corresponding specific features needed by each.

The following subsections describe frequently used concepts. Many of these concepts are described in the Base Definitions volume of IEEE Std 1003.1-200x. Utility and function description statements override these defaults when appropriate.

1.7.1.1 Process Attributes

The following process attributes, as described in the System Interfaces volume of IEEE Std 1003.1-200x, are assumed to be supported for all processes in this volume of IEEE Std 1003.1-200x:

Controlling Terminal	Real Group ID
Current Working Directory	Real User ID
Effective Group ID	Root Directory
Effective User ID	Saved Set-Group-ID
File Descriptors	Saved Set-User-ID
File Mode Creation Mask	Session Membership
Process Group ID	Supplementary Group IDs
Process ID	

A conforming implementation may include additional process attributes.

1.7.1.2 Concurrent Execution of Processes

The following functionality of the *fork*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x shall be available on all systems conforming to this volume of IEEE Std 1003.1-200x:

1. Independent processes shall be capable of executing independently without either process terminating.
2. A process shall be able to create a new process with all of the attributes referenced in Section 1.7.1.1, determined according to the semantics of a call to the *fork*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x followed by a call in the child process to one of the *exec* functions defined in the System Interfaces volume of IEEE Std 1003.1-200x.

105 *1.7.1.3 File Access Permissions*

106 The file access control mechanism described by the Base Definitions volume of |
107 IEEE Std 1003.1-200x, Section 4.4, File Access Permissions shall apply to all files on an |
108 implementation conforming to this volume of IEEE Std 1003.1-200x. |

109 *1.7.1.4 File Read, Write, and Creation*

110 If a file that does not exist is to be written, it shall be created as described below, unless the
111 utility description states otherwise.

112 When a file that does not exist is created, the following features defined in the System Interfaces
113 volume of IEEE Std 1003.1-200x shall apply unless the utility or function description states
114 otherwise:

115 1. The user ID of the file shall be set to the effective user ID of the calling process. |

116 2. The group ID of the file shall be set to the effective group ID of the calling process or the |
117 group ID of the directory in which the file is being created.

118 3. If the file is a regular file, the permission bits of the file shall be set to: |

119 S_IROTH | S_IWOTH | S_IRGRP | S_IWGRP | S_IRUSR | S_IWUSR

120 (see the description of *File Modes* in the Base Definitions volume of IEEE Std 1003.1-200x,
121 Chapter 13, Headers, **<sys/stat.h>**) except that the bits specified by the file mode creation
122 mask of the process shall be cleared. If the file is a directory, the permission bits shall be set |
123 to: |

124 S_IRWXU | S_IRWXG | S_IRWXO

125 except that the bits specified by the file mode creation mask of the process shall be cleared. |

126 4. The *st_atime*, *st_ctime*, and *st_mtime* fields of the file shall be updated as specified in the
127 System Interfaces volume of IEEE Std 1003.1-200x, Section 2.5, Standard I/O Streams.

128 5. If the file is a directory, it shall be an empty directory; otherwise, the file shall have length
129 zero.

130 6. If the file is a symbolic link, the effect shall be undefined unless the {POSIX2_SYMLINKS}
131 variable is in effect for the directory in which the symbolic link would be created.

132 7. Unless otherwise specified, the file created shall be a regular file.

133 When an attempt is made to create a file that already exists, the action shall depend on the type |
134 of the file the utility is trying to create and on the type of the existing file as shown in Table 1-1 |
135 (on page 2205). |

Table 1-1 Actions when Creating a File that Already Exists

Existing Type	New Type A	B	C	D	F	L	M	P	Q	R	S	T	Function Creating New
A *fattach*( )-ed STREAM	—	—	—	—	—	—	—	—	—	OF	—	—	*fattach*( )
B Block Special	—	—	—	—	—	—	—	—	—	OF	—	—	*mknod*( )**
C Character Special	—	—	—	—	—	—	—	—	—	OF	—	—	*mknod*( )**
D Directory	—	—	—	F	F	—	—	—	—	OD	—	—	*mkdir*( )
F FIFO Special File	—	—	—	F	F	—	—	—	—	O	—	—	*mkfifo*( )
L Symbolic Link	FL	FL	FL	FL	FL	FL	FL	FL	FL	FL	FL	FL	*symlink*( )
M Shared Memory	—	—	—	—	—	—	—	—	—	—	—	—	*shm_open*( )
P Semaphore	—	—	—	—	—	—	—	—	—	—	—	—	*sem_open*( )
Q Message Queue	—	—	—	—	—	—	—	—	—	—	—	—	*mq_open*( )
R Regular File	—	—	—	F	F	—	—	—	—	RF	—	—	*open*( )
S Socket	—	—	—	—	—	—	—	—	—	—	—	—	*bind*( )
T Typed Memory	—	—	—	—	—	—	—	—	—	—	—	—	*
None.	—	—	—	NF	NF	NF	—	—	—	CF	NF	—	

The following codes are used in Table 1-1:

CF Create a new file as defined in Section 1.7.1.4 (on page 2204), items 1 through 7.

F Fail. When attempting to create a directory or FIFO special file, and the existing file is a directory, FIFO special file, or regular file, the attempt shall fail and the utility shall either continue with its operation or exit immediately with a non-zero exit status, depending on the description of the utility.

FL Follow link. Unless otherwise specified, the symbolic links shall be followed as specified for pathname resolution, and the operation performed shall be as if the target of the symbolic link (after all resolution) had been named. If the target of the symbolic link does not exist, it shall be as if that nonexistent target had been named directly.

NF Create a new file as described by the appropriate function.

O Open FIFO. When attempting to create a regular file, and the existing file is a FIFO special file:

1. If the FIFO is not already open for reading, the attempt shall block until the FIFO is opened for reading.
2. Once the FIFO is open for reading, the utility shall open the FIFO for writing and continue with its operation.

OD The directory shall be opened.

OF The named file shall be opened with the consequences defined for that file type.

RF Regular file. When attempting to create a regular file, and the existing file is a regular file:

1. The user ID, group ID, and permission bits of the file shall not be changed.
2. The file shall be truncated to zero length.
3. The *st_ctime* and *st_mtime* fields shall be marked for update.

— The effect is implementation-defined unless specified by the utility description.

* There is no portable way to create a file of this type.

** Not portable.

When a file is to be appended, the file shall be opened in a manner equivalent to using the O_APPEND flag, without the O_TRUNC flag, in the *open*() function defined in the System Interfaces volume of IEEE Std 1003.1-200x.

When a file is to be read or written, the file shall be opened with an access mode corresponding to the operation to be performed. If file access permissions deny access, the requested operation shall fail.

1.7.1.5 File Removal

When a directory that is the root directory or current working directory of any process is removed, the effect is implementation-defined. If file access permissions deny access, the requested operation shall fail. Otherwise, when a file is removed:

1. Its directory entry shall be removed from the file system.
2. The link count of the file shall be decremented.
3. If the file is an empty directory (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.143, Empty Directory):
    a. If no process has the directory open, the space occupied by the directory shall be freed and the directory shall no longer be accessible.
    b. If one or more processes have the directory open, the directory contents shall be preserved until all references to the file have been closed.
4. If the file is a directory that is not empty, the *st_ctime* field shall be marked for update.
5. If the file is not a directory:
    a. If the link count becomes zero:
        i. If no process has the file open, the space occupied by the file shall be freed and the file shall no longer be accessible.
        ii. If one or more processes have the file open, the file contents shall be preserved until all references to the file have been closed.
    b. If the link count is not reduced to zero, the *st_ctime* field shall be marked for update.
6. The *st_ctime* and *st_mtime* fields of the containing directory shall be marked for update.

1.7.1.6 File Time Values

All files shall have the three time values described by the Base Definitions volume of IEEE Std 1003.1-200x, Section 4.7, File Times Update.

1.7.1.7 File Contents

When a reference is made to the contents of a file, *pathname*, this means the equivalent of all of the data placed in the space pointed to by *buf* when performing the *read*() function calls in the following operations defined in the System Interfaces volume of IEEE Std 1003.1-200x:

```
while (read (fildes, buf, nbytes) > 0)
    ;
```

If the file is indicated by a pathname *pathname*, the file descriptor shall be determined by the equivalent of the following operation defined in the System Interfaces volume of IEEE Std 1003.1-200x:

```
fildes = open (pathname, O_RDONLY);
```

The value of *nbytes* in the above sequence is unspecified; if the file is of a type where the data returned by *read*( ) would vary with different values, the value shall be one that results in the most data being returned.

If the *read*( ) function calls would return an error, it is unspecified whether the contents of the file are considered to include any data from offsets in the file beyond where the error would be returned.

1.7.1.8 Pathname Resolution

The pathname resolution algorithm, described by the Base Definitions volume of IEEE Std 1003.1-200x, Section 4.11, Pathname Resolution, shall be used by implementations conforming to this volume of IEEE Std 1003.1-200x; see also the Base Definitions volume of IEEE Std 1003.1-200x, Section 4.5, File Hierarchy.

1.7.1.9 Changing the Current Working Directory

When the current working directory (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.436, Working Directory) is to be changed, unless the utility or function description states otherwise, the operation shall succeed unless a call to the *chdir*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x would fail when invoked with the new working directory pathname as its argument.

1.7.1.10 Establish the Locale

The functionality of the *setlocale*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x shall be available on all systems conforming to this volume of IEEE Std 1003.1-200x; that is, utilities that require the capability of establishing an international operating environment shall be permitted to set the specified category of the international environment.

1.7.1.11 Actions Equivalent to Functions

Some utility descriptions specify that a utility performs actions equivalent to a function defined in the System Interfaces volume of IEEE Std 1003.1-200x. Such specifications require only that the external effects be equivalent, not that any effect within the utility and visible only to the utility be equivalent.

1.7.2 Concepts Derived from the ISO C Standard

Some of the standard utilities perform complex data manipulation using their own procedure and arithmetic languages, as defined in their EXTENDED DESCRIPTION or OPERANDS sections. Unless otherwise noted, the arithmetic and semantic concepts (precision, type conversion, control flow, and so on) shall be equivalent to those defined in the ISO C standard, as described in the following sections. Note that there is no requirement that the standard utilities be implemented in any particular programming language.

1.7.2.1 Arithmetic Precision and Operations

Integer variables and constants, including the values of operands and option-arguments, used by the standard utilities listed in this volume of IEEE Std 1003.1-200x shall be implemented as equivalent to the ISO C standard **signed long** data type; floating point shall be implemented as equivalent to the ISO C standard **double** type. Conversions between types shall be as described in the ISO C standard. All variables shall be initialized to zero if they are not otherwise assigned

by the input to the application.

Arithmetic operators and functions shall be implemented as equivalent to those in the cited ISO C standard section, as listed in Table 1-2 (on page 2209).

Table 1-2 ISO C Standard Operators and Functions

Operation	ISO C Standard Equivalent Reference
( )	Section 6.5.1, Primary Expressions
postfix ++ postfix --	Section 6.5.2, Postfix Operators
unary + unary - prefix ++ prefix -- ~ ! *sizeof*( )	Section 6.5.3, Unary Operators
* / %	Section 6.5.5, Multiplicative Operators
+ -	Section 6.5.6, Additive Operators
<< >>	Section 6.5.7, Bitwise Shift Operators
<, <= >, >=	Section 6.5.8, Relational Operators
= = !=	Section 6.5.9, Equality Operators
&	Section 6.5.10, Bitwise AND Operator
^	Section 6.5.11, Bitwise Exclusive OR Operator
|	Section 6.5.12, Bitwise Inclusive OR Operator
&&	Section 6.5.13, Logical AND Operator
| |	Section 6.5.14, Logical OR Operator
expr?*expr*:*expr*	Section 6.5.15, Conditional Operator
=, *=, / =, %=, +=, -= <<=, >>=, &=, ^=, |=	Section 6.5.16, Assignment Operators
if ( ) **if** ( ) ... **else** **switch** ( )	Section 6.8.4, Selection Statements
while ( ) **do** ... **while** ( ) **for** ( )	Section 6.8.5, Iteration Statements
goto **continue** **break** **return**	Section 6.8.6, Jump Statements

The evaluation of arithmetic expressions shall be equivalent to that described in Section 6.5, Expressions, of the ISO C standard.

306 *1.7.2.2 Mathematical Functions*

307 Any mathematical functions with the same names as those in the following sections of the ISO C
308 standard:

309 • Section 7.12, Mathematics, `<math.h>`

310 • Section 7.20.2, Pseudo-Random Sequence Generation Functions

311 shall be implemented to return the results equivalent to those returned from a call to the
312 corresponding function described in the ISO C standard.

313 1.8 Portability

314 Some of the utilities in the Shell and Utilities volume of IEEE Std 1003.1-200x and functions in
315 the System Interfaces volume of IEEE Std 1003.1-200x describe functionality that might not be
316 fully portable to systems meeting the requirements for POSIX conformance (see the Base
317 Definitions volume of IEEE Std 1003.1-200x, Chapter 2, Conformance).

318 Where optional, enhanced, or reduced functionality is specified, the text is shaded and a code in
319 the margin identifies the nature of the option, extension, or warning (see Section 1.8.1). For
320 maximum portability, an application should avoid such functionality.

321 Unless the primary task of a utility is to produce textual material on its standard output,
322 application developers should not rely on the format or content of any such material that may be
323 produced. Where the primary task *is* to provide such material, but the output format is
324 incompletely specified, the description is marked with the OF margin code and shading.
325 Application developers are warned not to expect that the output of such an interface on one
326 system is any guide to its behavior on another system.

327 1.8.1 Codes

328 Codes and their meanings are listed in the Base Definitions volume of IEEE Std 1003.1-200x, but
329 are repeated here for convenience:

330 ADV Advisory Information
331 The functionality described is optional. The functionality described is also an extension to the
332 ISO C standard.

333 Where applicable, functions are marked with the ADV margin legend in the SYNOPSIS section.
334 Where additional semantics apply to a function, the material is identified by use of the ADV
335 margin legend.

336 AIO Asynchronous Input and Output
337 The functionality described is optional. The functionality described is also an extension to the
338 ISO C standard.

339 Where applicable, functions are marked with the AIO margin legend in the SYNOPSIS section.
340 Where additional semantics apply to a function, the material is identified by use of the AIO
341 margin legend.

342 BAR Barriers
343 The functionality described is optional. The functionality described is also an extension to the
344 ISO C standard.

345 Where applicable, functions are marked with the BAR margin legend in the SYNOPSIS section.
346 Where additional semantics apply to a function, the material is identified by use of the BAR
347 margin legend.

BE Batch Environment Services and Utilities
The functionality described is optional.

Where applicable, utilities are marked with the BE margin legend in the SYNOPSIS section. Where additional semantics apply to a utility, the material is identified by use of the BE margin legend.

CD C-Language Development Utilities
The functionality described is optional.

Where applicable, utilities are marked with the CD margin legend in the SYNOPSIS section. Where additional semantics apply to a utility, the material is identified by use of the CD margin legend.

CPT Process CPU-Time Clocks
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the CPT margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the CPT margin legend.

CS Clock Selection
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the CS margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the CS margin legend.

CX Extension to the ISO C standard
The functionality described is an extension to the ISO C standard. Application writers may make use of an extension as it is supported on all IEEE Std 1003.1-200x-conforming systems.

With each function or header from the ISO C standard, a statement to the effect that ‘‘any conflict is unintentional’’ is included. That is intended to refer to a direct conflict. IEEE Std 1003.1-200x acts in part as a profile of the ISO C standard, and it may choose to further constrain behaviors allowed to vary by the ISO C standard. Such limitations are not considered conflicts.

FD FORTRAN Development Utilities
The functionality described is optional.

Where applicable, utilities are marked with the FD margin legend in the SYNOPSIS section. Where additional semantics apply to a utility, the material is identified by use of the FD margin legend.

FR FORTRAN Runtime Utilities
The functionality described is optional.

Where applicable, utilities are marked with the FR margin legend in the SYNOPSIS section. Where additional semantics apply to a utility, the material is identified by use of the FR margin legend.

FSC File Synchronization
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the FSC margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the FSC

margin legend.

IP6 IPV6
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the IP6 margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the IP6 margin legend.

MC1 Advisory Information and either Memory Mapped Files or Shared Memory Objects
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

This is a shorthand notation for combinations of multiple option codes.

Where applicable, functions are marked with the MC1 margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the MC1 margin legend.

Refer to the Base Definitions volume of IEEE Std 1003.1-200x, Section 1.5.2, Margin Code Notation.

MC2 Memory Mapped Files, Shared Memory Objects, or Memory Protection
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

This is a shorthand notation for combinations of multiple option codes.

Where applicable, functions are marked with the MC2 margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the MC2 margin legend.

Refer to the Base Definitions volume of IEEE Std 1003.1-200x, Section 1.5.2, Margin Code Notation.

MF Memory Mapped Files
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the MF margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the MF margin legend.

ML Process Memory Locking
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the ML margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the ML margin legend.

MLR Range Memory Locking
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the MLR margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the MLR margin legend.

MON **Monotonic Clock**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the MON margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the MON margin legend.

MPR **Memory Protection**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the MPR margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the MPR margin legend.

MSG **Message Passing**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the MSG margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the MSG margin legend.

MX **IEC 60559 Floating-Point Option**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the MX margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the MX margin legend.

OB **Obsolescent**
The functionality described may be withdrawn in a future version of this volume of IEEE Std 1003.1-200x. Strictly Conforming POSIX Applications and Strictly Conforming XSI Applications shall not use obsolescent features.

OF **Output Format Incompletely Specified**
The functionality described is an XSI extension. The format of the output produced by the utility is not fully specified. It is therefore not possible to post-process this output in a consistent fashion. Typical problems include unknown length of strings and unspecified field delimiters.

OH **Optional Header**
In the SYNOPSIS section of some interfaces in the System Interfaces volume of IEEE Std 1003.1-200x an included header is marked as in the following example:

OH
```
#include <sys/types.h>
#include <grp.h>
struct group *getgrnam(const char *name);
```

This indicates that the marked header is not required on XSI-conformant systems.

PIO **Prioritized Input and Output**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the PIO margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the PIO margin legend.

PS **Process Scheduling**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the PS margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the PS margin legend.

RS **Raw Sockets**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the RS margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the RS margin legend.

RTS **Realtime Signals Extension**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the RTS margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the RTS margin legend.

SD **Software Development Utilities**
The functionality described is optional.

Where applicable, utilities are marked with the SD margin legend in the SYNOPSIS section. Where additional semantics apply to a utility, the material is identified by use of the SD margin legend.

SEM **Semaphores**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the SEM margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the SEM margin legend.

SHM **Shared Memory Objects**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the SHM margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the SHM margin legend.

SIO **Synchronized Input and Output**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the SIO margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the SIO margin legend.

SPI **Spin Locks**
The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the SPI margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the SPI margin legend.

SPN Spawn

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the SPN margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the SPN margin legend.

SS Process Sporadic Server

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the SS margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the SS margin legend.

TCT Thread CPU-Time Clocks

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the TCT margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TCT margin legend.

TEF Trace Event Filter

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the TEF margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TEF margin legend.

THR Threads

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the THR margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the THR margin legend.

TMO Timeouts

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the TMO margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TMO margin legend.

TMR Timers

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the TMR margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TMR margin legend.

TPI Thread Priority Inheritance

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the TPI margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TPI margin legend.

TPP Thread Priority Protection

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the TPP margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TPP margin legend.

TPS Thread Execution Scheduling

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the TPS margin legend for the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TPS margin legend.

TRC Trace

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the TRC margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TRC margin legend.

TRI Trace Inherit

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the TRI margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TRI margin legend.

TRL Trace Log

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the TRL margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TRL margin legend.

TSA Thread Stack Address Attribute

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the TSA margin legend for the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TSA margin legend.

TSF Thread-Safe Functions

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the TSF margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TSF margin legend.

TSH Thread Process-Shared Synchronization

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the TSH margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TSH margin legend.

TSP Thread Sporadic Server

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the TSP margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TSP margin legend.

TSS Thread Stack Address Size

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the TSS margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TSS margin legend.

TYM Typed Memory Objects

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the TYM margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the TYM margin legend.

UP User Portability Utilities

The functionality described is optional.

Where applicable, utilities are marked with the UP margin legend in the SYNOPSIS section. Where additional semantics apply to a utility, the material is identified by use of the UP margin legend.

XSI Extension

The functionality described is an XSI extension. Functionality marked XSI is also an extension to the ISO C standard. Application writers may confidently make use of an extension on all systems supporting the X/Open System Interfaces Extension.

If an entire SYNOPSIS section is shaded and marked XSI, all the functionality described in that reference page is an extension. See the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.439, XSI.

XSR XSI STREAMS

The functionality described is optional. The functionality described is also an extension to the ISO C standard.

Where applicable, functions are marked with the XSR margin legend in the SYNOPSIS section. Where additional semantics apply to a function, the material is identified by use of the XSR margin legend.

1.9 Utility Limits

This section lists magnitude limitations imposed by a specific implementation. The braces notation, {LIMIT}, is used in this volume of IEEE Std 1003.1-200x to indicate these values, but the braces are not part of the name.

Table 1-3 Utility Limit Minimum Values

Name	Description	Value
{POSIX2_BC_BASE_MAX}	The maximum *obase* value allowed by the *bc* utility.	99
{POSIX2_BC_DIM_MAX}	The maximum number of elements permitted in an array by the *bc* utility.	2048
{POSIX2_BC_SCALE_MAX}	The maximum *scale* value allowed by the *bc* utility.	99
{POSIX2_BC_STRING_MAX}	The maximum length of a string constant accepted by the *bc* utility.	1000
{POSIX2_COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order** keyword in the locale definition file; see the **border_start** keyword in the Base Definitions volume of IEEE Std 1003.1-200x, Section 7.3.2, LC_COLLATE.	2
{POSIX2_EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the *expr* utility.	32
{POSIX2_LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as processing text files. The length includes room for the trailing newline.	2048
{POSIX2_RE_DUP_MAX}	The maximum number of repeated occurrences of a BRE permitted when using the interval notation \{*m*,*n*\}; see the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.3.6, BREs Matching Multiple Characters.	255
{POSIX2_VERSION}	This value indicates the version of the utilities in this volume of IEEE Std 1003.1-200x that are provided by the implementation. It changes with each published version.	200xxxL

The values specified in Table 1-3 represent the lowest values conforming implementations shall provide and, consequently, the largest values on which an application can rely without further enquiries, as described below. These values shall be accessible to applications via the *getconf* utility (see *getconf* (on page 2683)) and through the *sysconf*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x. The literal names shown in Table 1-3 apply only to the *getconf* utility; the high-level language binding describes the exact form of each name to be used by the interfaces in that binding.

Implementations may provide more liberal, or less restrictive, values than shown in Table 1-3. These possibly more liberal values are accessible using the symbols in Table 1-4 (on page 2219).

The *sysconf*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x or the *getconf* utility return the value of each symbol on each specific implementation. The value so retrieved is the largest, or most liberal, value that is available throughout the session lifetime, as determined at session creation. The literal names shown in the table apply only to the *getconf* utility; the high-level language binding describes the exact form of each name to be used by the interfaces in that binding.

All numeric limits defined by the System Interfaces volume of IEEE Std 1003.1-200x, such as {PATH_MAX}, shall also apply to this volume of IEEE Std 1003.1-200x. All the utilities defined by this volume of IEEE Std 1003.1-200x are implicitly limited by these values, unless otherwise noted in the utility descriptions.

It is not guaranteed that the application can actually reach the specified limit of an implementation in any given case, or at all, as a lack of virtual memory or other resources may prevent this. The limit value indicates only that the implementation does not specifically impose any arbitrary, more restrictive limit.

Table 1-4 Symbolic Utility Limits

Name	Description	Minimum Value
{BC_BASE_MAX}	The maximum *obase* value allowed by the *bc* utility.	{POSIX2_BC_BASE_MAX}
{BC_DIM_MAX}	The maximum number of elements permitted in an array by the *bc* utility.	{POSIX2_BC_DIM_MAX}
{BC_SCALE_MAX}	The maximum *scale* value allowed by the *bc* utility.	{POSIX2_BC_SCALE_MAX}
{BC_STRING_MAX}	The maximum length of a string constant accepted by the *bc* utility.	{POSIX2_BC_STRING_MAX}
{COLL_WEIGHTS_MAX}	The maximum number of weights that can be assigned to an entry of the *LC_COLLATE* **order** keyword in the locale definition file; see the **order_start** keyword in the Base Definitions volume of IEEE Std 1003.1-200x, Section 7.3.2, LC_COLLATE.	{POSIX2_COLL_WEIGHTS_MAX}
{EXPR_NEST_MAX}	The maximum number of expressions that can be nested within parentheses by the *expr* utility.	{POSIX2_EXPR_NEST_MAX}
{LINE_MAX}	Unless otherwise noted, the maximum length, in bytes, of the input line of a utility (either standard input or another file), when the utility is described as	{POSIX2_LINE_MAX}

Name	Description	Minimum Value
	processing text files. The length includes room for the trailing newline.	
{RE_DUP_MAX}	The maximum number of repeated occurrences of a BRE permitted when using the interval notation \{*m*,*n*\}; see the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.3.6, BREs Matching Multiple Characters.	{POSIX2_RE_DUP_MAX}

The following value may be a constant within an implementation or may vary from one pathname to another.

{POSIX2_SYMLINKS}
When referring to a directory, the system supports the creation of symbolic links within that directory; for non-directory files, the meaning of {POSIX2_SYMLINKS} is undefined.

1.10 Grammar Conventions

Portions of this volume of IEEE Std 1003.1-200x are expressed in terms of a special grammar notation. It is used to portray the complex syntax of certain program input. The grammar is based on the syntax used by the *yacc* utility. However, it does not represent fully functional *yacc* input, suitable for program use; the lexical processing and all semantic requirements are described only in textual form. The grammar is not based on source used in any traditional implementation and has not been tested with the semantic code that would normally be required to accompany it. Furthermore, there is no implication that the partial *yacc* code presented represents the most efficient, or only, means of supporting the complex syntax within the utility. Implementations may use other programming languages or algorithms, as long as the syntax supported is the same as that represented by the grammar.

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

- The identifiers for the reserved words of the language are shown with a leading capital letter. (These are terminals in the grammar; for example, **While**, **Case**.)
- The identifiers for terminals in the grammar are all named with uppercase letters and underscores; for example, **NEWLINE**, **ASSIGN_OP**, **NAME**.
- The identifiers for non-terminals are all lowercase.

1.11 Utility Description Defaults

This section describes all of the subsections used within the utility descriptions, including:

- Intended usage of the section
- Global defaults that affect all the standard utilities
- The meanings of notations used in this volume of IEEE Std 1003.1-200x that are specific to individual utility sections

NAME

This section gives the name or names of the utility and briefly states its purpose.

SYNOPSIS

The SYNOPSIS section summarizes the syntax of the calling sequence for the utility, including options, option-arguments, and operands. Standards for utility naming are described in the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines; for describing the utility's arguments in the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.1, Utility Argument Syntax.

DESCRIPTION

The DESCRIPTION section describes the actions of the utility. If the utility has a very complex set of subcommands or its own procedural language, an EXTENDED DESCRIPTION section is also provided. Most explanations of optional functionality are omitted here, as they are usually explained in the OPTIONS section.

As stated in Section 1.7.1.11 (on page 2207), some functions are described in terms of equivalent functionality. When specific functions are cited, the implementation shall provide equivalent functionality including side effects associated with successful execution of the function. The treatment of errors and intermediate results from the individual functions cited is generally not specified by this volume of IEEE Std 1003.1-200x. See the utility's EXIT STATUS and CONSEQUENCES OF ERRORS sections for all actions associated with errors encountered by the utility.

OPTIONS

The OPTIONS section describes the utility options and option-arguments, and how they modify the actions of the utility. Standard utilities that have options either fully comply with the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines or describe all deviations. Apparent disagreements between functionality descriptions in the OPTIONS and DESCRIPTION (or EXTENDED DESCRIPTION) sections are always resolved in favor of the OPTIONS section.

Each OPTIONS section that uses the phrase ''The ... utility shall conform to the Utility Syntax Guidelines ...'' refers only to the use of the utility as specified by this volume of IEEE Std 1003.1-200x; implementation extensions should also conform to the guidelines, but may allow exceptions for historical practice.

Unless otherwise stated in the utility description, when given an option unrecognized by the implementation, or when a required option-argument is not provided, standard utilities shall issue a diagnostic message to standard error and exit with a non-zero exit status.

All utilities in this volume of IEEE Std 1003.1-200x shall be capable of processing arguments using eight-bit transparency.

Default Behavior: When this section is listed as ''None.'', it means that the implementation need not support any options. Standard utilities that do not accept options, but that do accept operands, shall recognize "--" as a first argument to be

discarded.

The requirement for recognizing "−−" is because conforming applications need a way to shield their operands from any arbitrary options that the implementation may provide as an extension. For example, if the standard utility *foo* is listed as taking no options, and the application needed to give it a pathname with a leading hyphen, it could safely do it as:

```
foo −− −myfile
```

and avoid any problems with **−m** used as an extension.

OPERANDS

The OPERANDS section describes the utility operands, and how they affect the actions of the utility. Apparent disagreements between functionality descriptions in the OPERANDS and DESCRIPTION (or EXTENDED DESCRIPTION) sections shall be resolved in favor of the OPERANDS section.

If an operand naming a file can be specified as '−', which means to use the standard input instead of a named file, this is explicitly stated in this section. Unless otherwise stated, the use of multiple instances of '−' to mean standard input in a single command produces unspecified results.

Unless otherwise stated, the standard utilities that accept operands shall process those operands in the order specified in the command line.

Default Behavior: When this section is listed as ''None.'', it means that the implementation need not support any operands.

STDIN

The STDIN section describes the standard input of the utility. This section is frequently merely a reference to the following section, as many utilities treat standard input and input files in the same manner. Unless otherwise stated, all restrictions described in the INPUT FILES section shall apply to this section as well.

Use of a terminal for standard input can cause any of the standard utilities that read standard input to stop when used in the background. For this reason, applications should not use interactive features in scripts to be placed in the background.

The specified standard input format of the standard utilities shall not depend on the existence or value of the environment variables defined in this volume of IEEE Std 1003.1-200x, except as provided by this volume of IEEE Std 1003.1-200x.

Default Behavior: When this section is listed as ''Not used.'', it means that the standard input shall not be read when the utility is used as described by this volume of IEEE Std 1003.1-200x.

INPUT FILES

The INPUT FILES section describes the files, other than the standard input, used as input by the utility. It includes files named as operands and option-arguments as well as other files that are referred to, such as start-up and initialization files, databases, and so on. Commonly-used files are generally described in one place and cross-referenced by other utilities.

All utilities in this volume of IEEE Std 1003.1-200x shall be capable of processing input files using eight-bit transparency.

When a standard utility reads a seekable input file and terminates without an error before it reaches end-of-file, the utility shall ensure that the file offset in the open file

description is properly positioned just past the last byte processed by the utility. For files that are not seekable, the state of the file offset in the open file description for that file is unspecified. A conforming application shall not assume that the following three commands are equivalent:

```
tail –n +2 file
(sed –n 1q; cat) < file
cat file | (sed –n 1q; cat)
```

The second command is equivalent to the first only when the file is seekable. The third command leaves the file offset in the open file description in an unspecified state. Other utilities, such as *head*, *read*, and *sh*, have similar properties.

Some of the standard utilities, such as filters, process input files a line or a block at a time and have no restrictions on the maximum input file size. Some utilities may have size limitations that are not as obvious as file space or memory limitations. Such limitations should reflect resource limitations of some sort, not arbitrary limits set by implementors. Implementations shall document those utilities that are limited by constraints other than file system space, available memory, and other limits specifically cited by this volume of IEEE Std 1003.1-200x, and identify what the constraint is and indicate a way of estimating when the constraint would be reached. Similarly, some utilities descend the directory tree (recursively). Implementations shall also document any limits that they may have in descending the directory tree that are beyond limits cited by this volume of IEEE Std 1003.1-200x.

When an input file is described as a *text file*, the utility produces undefined results if given input that is not from a text file, unless otherwise stated. Some utilities (for example, *make*, *read*, *sh*) allow for continued input lines using an escaped <newline> convention; unless otherwise stated, the utility need not be able to accumulate more than {LINE_MAX} bytes from a set of multiple, continued input lines. Thus, for a conforming application the total of all the continued lines in a set cannot exceed {LINE_MAX}. If a utility using the escaped <newline> convention detects an end-of-file condition immediately after an escaped <newline>, the results are unspecified.

Record formats are described in a notation similar to that used by the C-language function, *printf*( ). See the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation for a description of this notation. The format description is intended to be sufficiently rigorous to allow other applications to generate these input files. However, since <blank>s can legitimately be included in some of the fields described by the standard utilities, particularly in locales other than the POSIX locale, this intent is not always realized.

Default Behavior: When this section is listed as ‘‘None.’’, it means that no input files are required to be supplied when the utility is used as described by this volume of IEEE Std 1003.1-200x.

ENVIRONMENT VARIABLES

The ENVIRONMENT VARIABLES section lists what variables affect the utility’s execution.

The entire manner in which environment variables described in this volume of IEEE Std 1003.1-200x affect the behavior of each utility is described in the ENVIRONMENT VARIABLES section for that utility, in conjunction with the global XSI effects of the *LANG*, *LC_ALL*, and *NLSPATH* environment variables described in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables. The existence or value of environment variables described in this volume of

IEEE Std 1003.1-200x shall not otherwise affect the specified behavior of the standard utilities. Any effects of the existence or value of environment variables not described by this volume of IEEE Std 1003.1-200x upon the standard utilities are unspecified.

For those standard utilities that use environment variables as a means for selecting a utility to execute (such as *CC* in *make*), the string provided to the utility is subjected to the path search described for *PATH* in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

All utilities in this volume of IEEE Std 1003.1-200x shall be capable of processing environment variable names and values using eight-bit transparency.

Default Behavior: When this section is listed as ''None.'', it means that the behavior of the utility is not directly affected by environment variables described by this volume of IEEE Std 1003.1-200x when the utility is used as described by this volume of IEEE Std 1003.1-200x.

ASYNCHRONOUS EVENTS

The ASYNCHRONOUS EVENTS section lists how the utility reacts to such events as signals and what signals are caught.

Default Behavior: When this section is listed as ''Default.'', or it refers to ''the standard action for all other signals; see Section 1.11 (on page 2221)'' it means that the action taken as a result of the signal shall be one of the following:

1. The action shall be that inherited from the parent according to the rules of inheritance of signal actions defined in the System Interfaces volume of IEEE Std 1003.1-200x.
2. When no action has been taken to change the default, the default action shall be that specified by the System Interfaces volume of IEEE Std 1003.1-200x.
3. The result of the utility's execution is as if default actions had been taken.

A utility is permitted to catch a signal, perform some additional processing (such as deleting temporary files), restore the default signal action (or action inherited from the parent process), and resignal itself.

STDOUT

The STDOUT section completely describes the standard output of the utility. This section is frequently merely a reference to the following section, OUTPUT FILES, because many utilities treat standard output and output files in the same manner.

Use of a terminal for standard output may cause any of the standard utilities that write standard output to stop when used in the background. For this reason, applications should not use interactive features in scripts to be placed in the background.

Record formats are described in a notation similar to that used by the C-language function, *printf*( ). See the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation for a description of this notation.

The specified standard output of the standard utilities shall not depend on the existence or value of the environment variables defined in this volume of IEEE Std 1003.1-200x, except as provided by this volume of IEEE Std 1003.1-200x.

Some of the standard utilities describe their output using the verb *display*, defined in the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.132, Display. Output described in the STDOUT sections 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.

Default Behavior: When this section is listed as ''Not used.'', it means that the standard output shall not be written when the utility is used as described by this volume of IEEE Std 1003.1-200x.

STDERR

The STDERR section describes the standard error output of the utility. Only those messages that are purposely sent by the utility are described.

Use of a terminal for standard error may cause any of the standard utilities that write standard error output to stop when used in the background. For this reason, applications should not use interactive features in scripts to be placed in the background.

The format of diagnostic messages for most utilities is unspecified, but the language and cultural conventions of diagnostic and informative messages whose format is unspecified by this volume of IEEE Std 1003.1-200x should be affected by the setting of XSI *LC_MESSAGES* and *NLSPATH*.

The specified standard error output of standard utilities shall not depend on the existence or value of the environment variables defined in this volume of IEEE Std 1003.1-200x, except as provided by this volume of IEEE Std 1003.1-200x.

Default Behavior: When this section is listed as ''Used only for diagnostic messages.'', it means that, unless otherwise stated, the diagnostic messages shall be sent to the standard error only when the exit status is non-zero and the utility is used as described by this volume of IEEE Std 1003.1-200x.

When this section is listed as ''Not used.'', it means that the standard error shall not be used when the utility is used as described in this volume of IEEE Std 1003.1-200x.

OUTPUT FILES

The OUTPUT FILES section completely describes the files created or modified by the utility. Temporary or system files that are created for internal usage by this utility or other parts of the implementation (for example, spool, log, and audit files) are not described in this, or any, section. The utilities creating such files and the names of such files are unspecified. If applications are written to use temporary or intermediate files, they should use the *TMPDIR* environment variable, if it is set and represents an accessible directory, to select the location of temporary files.

Implementations shall ensure that temporary files, when used by the standard utilities, are named so that different utilities or multiple instances of the same utility can operate simultaneously without regard to their working directories, or any other process characteristic other than process ID. There are two exceptions to this rule:

1. Resources for temporary files other than the name space (for example, disk space, available directory entries, or number of processes allowed) are not guaranteed.
2. Certain standard utilities generate output files that are intended as input for other utilities (for example, *lex* generates **lex.yy.c**), and these cannot have unique names. These cases are explicitly identified in the descriptions of the respective utilities.

Any temporary file created by the implementation shall be removed by the implementation upon a utility's successful exit, exit because of errors, or before termination by any of the SIGHUP, SIGINT, or SIGTERM signals, unless specified otherwise by the utility description.

Receipt of the SIGQUIT signal should generally cause termination (unless in some debugging mode) that would bypass any attempted recovery actions.

Record formats are described in a notation similar to that used by the C-language function, *printf*( ); see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation for a description of this notation.

Default Behavior: When this section is listed as ''None.'', it means that no files are created or modified as a consequence of direct action on the part of the utility when the utility is used as described by this volume of IEEE Std 1003.1-200x. However, the utility may create or modify system files, such as log files, that are outside the utility's normal execution environment.

EXTENDED DESCRIPTION

The EXTENDED DESCRIPTION section provides a place for describing the actions of very complicated utilities, such as text editors or language processors, which typically have elaborate command languages.

Default Behavior: When this section is listed as ''None.'', no further description is necessary.

EXIT STATUS

The EXIT STATUS section describes the values the utility shall return to the calling program, or shell, and the conditions that cause these values to be returned. Usually, utilities return zero for successful completion and values greater than zero for various error conditions. If specific numeric values are listed in this section, the system shall use those values for the errors described. In some cases, status values are listed more loosely, such as >0. A strictly conforming application shall not rely on any specific value in the range shown and shall be prepared to receive any value in the range.

For example, a utility may list zero as a successful return, 1 as a failure for a specific reason, and >1 as ''an error occurred''. In this case, unspecified conditions may cause a 2 or 3, or other value, to be returned. A conforming application should be written so that it tests for successful exit status values (zero in this case), rather than relying upon the single specific error value listed in this volume of IEEE Std 1003.1-200x. In that way, it has maximum portability, even on implementations with extensions.

Unspecified error conditions may be represented by specific values not listed in this volume of IEEE Std 1003.1-200x.

CONSEQUENCES OF ERRORS

The CONSEQUENCES OF ERRORS section describes the effects on the environment, file systems, process state, and so on, when error conditions occur. It does not describe error messages produced or exit status values used.

The many reasons for failure of a utility are generally not specified by the utility descriptions. Utilities may terminate prematurely if they encounter: invalid usage of options, arguments, or environment variables; invalid usage of the complex syntaxes expressed in EXTENDED DESCRIPTION sections; difficulties accessing, creating, reading, or writing files; or difficulties associated with the privileges of the process.

The following shall apply to each utility, unless otherwise stated:

- If the requested action cannot be performed on an operand representing a file, directory, user, process, and so on, the utility shall issue a diagnostic message to standard error and continue processing the next operand in sequence, but the final exit status shall be returned as non-zero.

For a utility that recursively traverses a file hierarchy (such as *find* or *chown* –**R**), if the requested action cannot be performed on a file or directory encountered in the hierarchy, the utility shall issue a diagnostic message to standard error and continue processing the remaining files in the hierarchy, but the final exit status shall be returned as non-zero.

- If the requested action characterized by an option or option-argument cannot be performed, the utility shall issue a diagnostic message to standard error and the exit status returned shall be non-zero.
- When an unrecoverable error condition is encountered, the utility shall exit with a non-zero exit status.
- A diagnostic message shall be written to standard error whenever an error condition occurs.

When a utility encounters an error condition several actions are possible, depending on the severity of the error and the state of the utility. Included in the possible actions of various utilities are: deletion of temporary or intermediate work files; deletion of incomplete files; validity checking of the file system or directory.

Default Behavior: When this section is listed as ''Default.'', it means that any changes to the environment are unspecified.

APPLICATION USAGE

This section is non-normative.

The APPLICATION USAGE section gives advice to the application programmer or user about the way the utility should be used.

EXAMPLES

This section is non-normative.

The EXAMPLES section gives one or more examples of usage, where appropriate. In the event of conflict between an example and a normative part of the specification, the normative material is to be taken as correct.

In all examples, quoting has been used, showing how sample commands (utility names combined with arguments) could be passed correctly to a shell (see *sh*) or as a string to the *system*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x. Such quoting would not be used if the utility is invoked using one of the *exec* functions defined in the System Interfaces volume of IEEE Std 1003.1-200x.

RATIONALE

This section is non-normative.

This section contains historical information concerning the contents of this volume of IEEE Std 1003.1-200x and why features were included or discarded by the standard developers.

FUTURE DIRECTIONS

This section is non-normative.

The FUTURE DIRECTIONS section should be used as a guide to current thinking; there is not necessarily a commitment to implement all of these future directions in their entirety.

SEE ALSO

This section is non-normative.

The SEE ALSO section lists related entries.

CHANGE HISTORY

This section is non-normative.

The CHANGE HISTORY section shows the derivation of the description used by this volume of IEEE Std 1003.1-200x and lists the functional differences between Issues 4 and 6.

Certain of the standard utilities describe how they can invoke other utilities or applications, such as by passing a command string to the command interpreter. The external influences (STDIN, ENVIRONMENT VARIABLES, and so on) and external effects (STDOUT, CONSEQUENCES OF ERRORS, and so on) of such invoked utilities are not described in the section concerning the standard utility that invokes them.

1.12 Considerations for Utilities in Support of Files of Arbitrary Size

The following utilities support files of any size up to the maximum that can be created by the implementation. This support includes correct writing of file size-related values (such as file sizes and offsets, line numbers, and block counts) and correct interpretation of command line arguments that contain such values.

basename	Return non-directory portion of pathname.
cat	Concatenate and print files.
cd	Change working directory.
chgrp	Change file group ownership.
chmod	Change file modes.
chown	Change file ownership.
cksum	Write file checksums and sizes.
cmp	Compare two files.
cp	Copy files.
dd	Convert and copy a file.
df	Report free disk space.
dirname	Return directory portion of pathname.
du	Estimate file space usage.
find	Find files.
ln	Link files.
ls	List directory contents.
mkdir	Make directories.
mv	Move files.
pathchk	Check pathnames.
pwd	Return working directory name.

rm	Remove directory entries.
rmdir	Remove directories.
sh	Shell, the standard command language interpreter.
sum	Print checksum and block or byte count of a file.
test	Evaluate expression.
touch	Change file access and modification times.
ulimit	Set or report file size limit.

Exceptions to the requirement that utilities support files of any size up to the maximum are as follows:

1. Uses of files as command scripts, or for configuration or control, are exempt. For example, it is not required that *sh* be able to read an arbitrarily large **.profile**.
2. Shell input and output redirection are exempt. For example, it is not required that the redirections *sum* < *file* or *echo foo* > *file* succeed for an arbitrarily large existing file.

1.13 Built-In Utilities

Any of the standard utilities may be implemented as *regular built-in* utilities within the command language interpreter. This is usually done to increase the performance of frequently used utilities or to achieve functionality that would be more difficult in a separate environment. The utilities named in Table 1-5 are frequently provided in built-in form. All of the utilities named in the table have special properties in terms of command search order within the shell, as described in Section 2.9.1.1 (on page 2249).

Table 1-5 Regular Built-in Utilities

alias	*false*	*jobs*	*true*
bg	*fc*	*kill*	*umask*
cd	*fg*	*newgrp*	*unalias*
command	*getopts*	*read*	*wait*

However, all of the standard utilities, including the regular built-ins in the table, but not the special built-ins described in Section 2.14 (on page 2266), shall be implemented in a manner so that they can be accessed via the *exec* family of functions as defined in the System Interfaces volume of IEEE Std 1003.1-200x and can be invoked directly by those standard utilities that require it (*env*, *find*, *nice*, *nohup*, *time*, *xargs*).

Since *exec*-able versions shall be provided for all utilities except for those listed in Section 2.14 (on page 2266), an application running on a system that conforms to IEEE Std 1003.1-200x can use the *exec* family of functions, in addition to the shell command interface provided by the *system*( ) and *popen*( ) functions, to execute any of these utilities.

1182

Chapter 2

Shell Command Language

This chapter contains the definition of the Shell Command Language.

2.1 Shell Introduction

The shell is a command language interpreter. This chapter describes the syntax of that command language as it is used by the *sh* utility and the *system*( ) and *popen*( ) functions defined in the System Interfaces volume of IEEE Std 1003.1-200x.

The shell operates according to the following general overview of operations. The specific details are included in the cited sections of this chapter.

1. The shell reads its input from a file (see *sh*), from the **–c** option or from the *system*( ) and *popen*( ) functions defined in the System Interfaces volume of IEEE Std 1003.1-200x. If the first line of a file of shell commands starts with the characters "#!", the results are unspecified.
2. The shell breaks the input into tokens: words and operators; see Section 2.3 (on page 2233).
3. The shell parses the input into simple commands (see Section 2.9.1 (on page 2248)) and compound commands (see Section 2.9.4 (on page 2253)).
4. The shell performs various expansions (separately) on different parts of each command, resulting in a list of pathnames and fields to be treated as a command and arguments; see Section 2.6 (on page 2238).
5. The shell performs redirection (see Section 2.7 (on page 2244)) and removes redirection operators and their operands from the parameter list.
6. The shell executes a function (see Section 2.9.5 (on page 2256)), built-in (see Section 2.14 (on page 2266)), executable file, or script, giving the names of the arguments as positional parameters numbered 1 to *n*, and the name of the command (or in the case of a function within a script, the name of the script) as the positional parameter numbered 0 (see Section 2.9.1.1 (on page 2249)).
7. The shell optionally waits for the command to complete and collects the exit status (see Section 2.8.2 (on page 2248)).

2.2 Quoting

Quoting is used to remove the special meaning of certain characters or words to the shell. Quoting can be used to preserve the literal meaning of the special characters in the next paragraph, prevent reserved words from being recognized as such, and prevent parameter expansion and command substitution within here-document processing (see Section 2.7.4 (on page 2246)).

The application shall quote the following characters if they are to represent themselves:

```
|  &  ;  <  >  (  )  $  '  \  "  '  <space>  <tab>  <newline>
```

and the following may need to be quoted under certain circumstances. That is, these characters may be special depending on conditions described elsewhere in this volume of IEEE Std 1003.1-200x:

```
*  ?  [  #  ~  =  %
```

The various quoting mechanisms are the escape character, single-quotes, and double-quotes. The here-document represents another form of quoting; see Section 2.7.4 (on page 2246).

2.2.1 Escape Character (Backslash)

A backslash that is not quoted shall preserve the literal value of the following character, with the exception of a <newline>. If a <newline> follows the backslash, the shell shall interpret this as line continuation. The backslash and <newline>s shall be removed before splitting the input into tokens. Since the escaped <newline> is removed entirely from the input and is not replaced by any white space, it cannot serve as a token separator.

2.2.2 Single-Quotes

Enclosing characters in single-quotes (`''`) shall preserve the literal value of each character within the single-quotes. A single-quote cannot occur within single-quotes.

2.2.3 Double-Quotes

Enclosing characters in double-quotes (`""`) shall preserve the literal value of all characters within the double-quotes, with the exception of the characters dollar sign, backquote, and backslash, as follows:

`$` The dollar sign shall retain its special meaning introducing parameter expansion (see Section 2.6.2 (on page 2239)), a form of command substitution (see Section 2.6.3 (on page 2242)), and arithmetic expansion (see Section 2.6.4 (on page 2243)).

The input characters within the quoted string that are also enclosed between `"$("` and the matching `')'` is not affected by the double-quotes, but rather shall define that command whose output replaces the `"$(...)"` when the word is expanded. The tokenizing rules in Section 2.3 (on page 2233), not including the alias substitutions in Section 2.3.1 (on page 2234), shall be applied recursively to find the matching `')'`.

Within the string of characters from an enclosed `"${"` to the matching `'}'`, an even number of unescaped double-quotes or single-quotes, if any, shall occur. A preceding backslash character shall be used to escape a literal `'{'` or `'}'`. The rule in Section 2.6.2 (on page 2239) shall be used to determine the matching `'}'`.

`` ` `` The backquote shall retain its special meaning introducing the other form of command substitution (see Section 2.6.3 (on page 2242)). The portion of the quoted string from the initial backquote and the characters up to the next backquote that is not preceded by a

backslash, having escape characters removed, defines that command whose output replaces "`...`" when the word is expanded. Either of the following cases produces undefined results:

- A single-quoted or double-quoted string that begins, but does not end, within the "`...`" sequence
- A "`...`" sequence that begins, but does not end, within the same double-quoted string

\ The backslash shall retain its special meaning as an escape character (see Section 2.2.1 (on page 2232)) only when followed by one of the following characters when considered special:

```
$   `   "   \   <newline>
```

The application shall ensure that a double-quote is preceded by a backslash to be included within double-quotes. The parameter '@' has special meaning inside double-quotes and is described in Section 2.5.2 (on page 2235).

2.3 Token Recognition

The shell shall read its input in terms of lines from a file, from a terminal in the case of an interactive shell, or from a string in the case of *sh* **–c** or *system*( ). The input lines can be of unlimited length. These lines shall be parsed using two major modes: ordinary token recognition and processing of here-documents.

When an **io_here** token has been recognized by the grammar (see Section 2.10 (on page 2257)), one or more of the subsequent lines immediately following the next **NEWLINE** token form the body of one or more here-documents and shall be parsed according to the rules of Section 2.7.4 (on page 2246).

When it is not processing an **io_here**, the shell shall break its input into tokens by applying the first applicable rule below to the next character in its input. The token shall be from the current position in the input until a token is delimited according to one of the rules below; the characters forming the token are exactly those in the input, including any quoting characters. If it is indicated that a token is delimited, and no characters have been included in a token, processing shall continue until an actual token is delimited.

1. If the end of input is recognized, the current token shall be delimited. If there is no current token, the end-of-input indicator shall be returned as the token.
2. If the previous character was used as part of an operator and the current character is not quoted and can be used with the current characters to form an operator, it shall be used as part of that (operator) token.
3. If the previous character was used as part of an operator and the current character cannot be used with the current characters to form an operator, the operator containing the previous character shall be delimited.
4. If the current character is backslash, single-quote, or double-quote ('\\', '\'', or '"') and it is not quoted, it shall affect quoting for subsequent characters up to the end of the quoted text. The rules for quoting are as described in Section 2.2 (on page 2232). During token recognition no substitutions shall be actually performed, and the result token shall contain exactly the characters that appear in the input (except for <newline> joining), unmodified, including any embedded or enclosing quotes or substitution operators, between the quote mark and the end of the quoted text. The token shall not be delimited by the end of the quoted field.

5. If the current character is an unquoted '$' or '`', the shell shall identify the start of any candidates for parameter expansion (Section 2.6.2 (on page 2239)), command substitution (Section 2.6.3 (on page 2242)), or arithmetic expansion (Section 2.6.4 (on page 2243)) from their introductory unquoted character sequences: '$' or "${", "$(" or '`', and "$((", respectively. The shell shall read sufficient input to determine the end of the unit to be expanded (as explained in the cited sections). While processing the characters, if instances of expansions or quoting are found nested within the substitution, the shell shall recursively process them in the manner specified for the construct that is found. The characters found from the beginning of the substitution to its end, allowing for any recursion necessary to recognize embedded constructs, shall be included unmodified in the result token, including any embedded or enclosing substitution operators or quotes. The token shall not be delimited by the end of the substitution.

6. If the current character is not quoted and can be used as the first character of a new operator, the current token (if any) shall be delimited. The current character shall be used as the beginning of the next (operator) token.

7. If the current character is an unquoted <newline>, the current token shall be delimited.

8. If the current character is an unquoted <blank>, any token containing the previous character is delimited and the current character shall be discarded.

9. If the previous character was part of a word, the current character shall be appended to that word.

10. If the current character is a '#', it and all subsequent characters up to, but excluding, the next <newline> shall be discarded as a comment. The <newline> that ends the line is not considered part of the comment.

11. The current character is used as the start of a new word.

Once a token is delimited, it is categorized as required by the grammar in Section 2.10 (on page 2257).

2.3.1 Alias Substitution

UP XSI The processing of aliases shall be supported on all XSI-conformant systems or if the system supports the User Portability Utilities option (and the rest of this section is not further shaded for these options).

After a token has been delimited, but before applying the grammatical rules in Section 2.10 (on page 2257), a resulting word that is identified to be the command name word of a simple command shall be examined to determine whether 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 the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.10, Alias Name) shall be one that has been defined by the *alias* utility and not subsequently undefined using *unalias*. 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 volume of IEEE Std 1003.1-200x, alias definitions shall not be inherited by separate invocations of the shell or by the utility execution environments invoked by the shell; see Section 2.12 (on page 2263).

2.4 Reserved Words

Reserved words are words that have special meaning to the shell; see Section 2.9 (on page 2248). The following words shall be recognized as reserved words:

!	**do**	**esac**	**in**
{	**done**	**fi**	**then**
}	**elif**	**for**	**until**
case	**else**	**if**	**while**

This recognition shall only occur when none of the characters is quoted and when the word is used as:

- The first word of a command
- The first word following one of the reserved words other than **case**, **for**, or **in**
- The third word in a **case** or **for** command (only **in** is valid in this case)

See the grammar in Section 2.10 (on page 2257).

The following words may be recognized as reserved words on some implementations (when none of the characters are quoted), causing unspecified results:

[[	**]]**	**function**	**select**

Words that are the concatenation of a name and a colon (`':'`) are reserved; their use produces unspecified results.

2.5 Parameters and Variables

A parameter can be denoted by a name, a number, or one of the special characters listed in Section 2.5.2. A variable is a parameter denoted by a name.

A parameter is set if it has an assigned value (null is a valid value). Once a variable is set, it can only be unset by using the *unset* special built-in command.

2.5.1 Positional Parameters

A positional parameter is a parameter denoted by the decimal value represented by one or more digits, other than the single digit 0. The digits denoting the positional parameters shall always be interpreted as a decimal value, even if there is a leading zero. When a positional parameter with more than one digit is specified, the application shall enclose the digits in braces (see Section 2.6.2 (on page 2239)). Positional parameters are initially assigned when the shell is invoked (see *sh*), temporarily replaced when a shell function is invoked (see Section 2.9.5 (on page 2256)), and can be reassigned with the *set* special built-in command.

2.5.2 Special Parameters

Listed below are the special parameters and the values to which they shall expand. Only the values of the special parameters are listed; see Section 2.6 (on page 2238) for a detailed summary of all the stages involved in expanding words.

@ Expands to the positional parameters, starting from one. When the expansion occurs within double-quotes, and where field splitting (see Section 2.6.5 (on page 2243)) is performed, each positional parameter shall expand as a separate field, with the provision that the expansion of the first parameter shall still be joined with the beginning part of the original word (assuming that the expanded parameter was embedded within a word), and the

expansion of the last parameter shall still be joined with the last part of the original word. If there are no positional parameters, the expansion of '@' shall generate zero fields, even when '@' is double-quoted.

* Expands to the positional parameters, starting from one. When the expansion occurs within a double-quoted string (see Section 2.2.3 (on page 2232)), it shall expand to a single field with the value of each parameter separated by the first character of the *IFS* variable, or by a <space> if *IFS* is unset. If *IFS* is set to a null string, this is not equivalent to unsetting it; its first character does not exist, so the parameter values are concatenated.

\# Expands to the decimal number of positional parameters. The command name (parameter 0) shall not be counted in the number given by '#' because it is a special parameter, not a positional parameter.

? Expands to the decimal exit status of the most recent pipeline (see Section 2.9.2 (on page 2250)).

– (Hyphen.) Expands to the current option flags (the single-letter option names concatenated into a string) as specified on invocation by the *set* special built-in command or implicitly by the shell.

$ Expands to the decimal process ID of the invoked shell. In a subshell (see Section 2.12 (on page 2263)), '$' shall expand to the same value as that of the current shell.

! Expands to the decimal process ID of the most recent background command (see Section 2.9.3 (on page 2251)) executed from the current shell. (For example, background commands executed from subshells do not affect the value of "$!" in the current shell environment.) For a pipeline, the process ID is that of the last command in the pipeline.

0 (Zero.) Expands to the name of the shell or shell script. See *sh* (on page 3048) for a detailed description of how this name is derived.

See the description of the *IFS* variable in Section 2.5.3.

2.5.3 Shell Variables

Variables shall be initialized from the environment (as defined by the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables and the *exec* function in the System Interfaces volume of IEEE Std 1003.1-200x) and can be given new values with variable assignment commands. If a variable is initialized from the environment, it shall be marked for export immediately; see the *export* special built-in. New variables can be defined and initialized with variable assignments, with the *read* or *getopts* utilities, with the *name* parameter in a **for** loop, with the ${*name*=*word*} expansion, or with other mechanisms provided as implementation extensions.

The following variables shall affect the execution of the shell.

UP XSI *ENV* The processing of the *ENV* shell variable shall be supported on all XSI-conformant systems or if the system supports the User Portability Utilities option.

This variable, when and only when an interactive shell is invoked, shall be subjected to parameter expansion (see Section 2.6.2 (on page 2239)) by the shell and the resulting value shall be used as a pathname of a file containing shell commands to execute in the current environment. The file need not be executable. If the expanded value of *ENV* is not an absolute 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.

HOME The pathname of the user's home directory. The contents of *HOME* are used in tilde expansion (see Section 2.6.1 (on page 2239)).

IFS (Input Field Separators.) A string treated as a list of characters that is used for field splitting and to split lines into fields with the *read* command. If *IFS* is not set, the shell shall behave as if the value of *IFS* is <space>, <tab>, and <newline>; see Section 2.6.5 (on page 2243).

LANG The default value for the internationalization variables that are unset or null. If *LANG* is unset or null, the corresponding value from the implementation-defined default locale is used. If any of the internationalization variables contains an invalid setting, the utility behaves as if none of the variables had been defined.

LC_ALL The default value for the *LC_** variables, as described in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

LC_COLLATE Determine the behavior of range expressions, equivalence classes, and multi-character collating elements within pattern matching.

LC_CTYPE Determine the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters), which characters are defined as letters (character class **alpha**) and <blank>s (character class **blank**), and the behavior of character classes within pattern matching. Changing the value of *LC_CTYPE* after the shell has started shall not affect the lexical processing of shell commands in the current shell execution environment or its subshells. Invoking a shell script or performing *exec sh* subjects the new shell to the changes in *LC_CTYPE*.

LC_MESSAGES Determine the language in which messages should be written.

LINENO Set by the shell to a decimal number representing the current sequential line number (numbered starting with 1) within a script or function before it executes each command. 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 volume of IEEE Std 1003.1-200x specifies the effects of the variable only for systems supporting the User Portability Utilities option.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

PATH A string formatted as described in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables, used to effect command interpretation; see Section 2.9.1.1 (on page 2249).

PPID Set by the shell to the decimal process ID of the process that invoked this shell. In a subshell (see Section 2.12 (on page 2263)), *PPID* shall be set to the same value as that of the parent of the current shell. For example, *echo*$*PPID* and (*echo*$*PPID*) would produce the same value. This volume of IEEE Std 1003.1-200x specifies the effects of the variable only for systems supporting the User Portability Utilities option.

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 "$ ". For users who have specific additional implementation-defined privileges, the default may be another, implementation-defined value. The shell shall replace each instance of the

character '!' in *PS1* with the history file number of the next command to be typed. Escaping the '!' with another '!' (that is, "!!") shall place the literal character '!' in the prompt. This volume of IEEE Std 1003.1-200x specifies the effects of the variable only for systems supporting the User Portability Utilities option.

PS2 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 is "> ". This volume of IEEE Std 1003.1-200x specifies the effects of the variable only for systems supporting the User Portability Utilities option.

PS4 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 is "+ ". This volume of IEEE Std 1003.1-200x specifies the effects of the variable only for systems supporting the User Portability Utilities option.

PWD Set by the shell to be an absolute pathname of the current working directory, containing no components of type symbolic link, no components that are dot, and no components that are dot-dot when the shell is initialized. If an application sets or unsets the value of *PWD*, the behaviors of the *cd* and *pwd* utilities are unspecified.

2.6 Word Expansions

This section describes the various expansions that are performed on words. Not all expansions are performed on every word, as explained in the following sections.

Tilde expansions, parameter expansions, command substitutions, arithmetic expansions, and quote removals that occur within a single word expand to a single field. It is only field splitting or pathname expansion that can create multiple fields from a single word. The single exception to this rule is the expansion of the special parameter '@' within double-quotes, as described in Section 2.5.2 (on page 2235).

The order of word expansion shall be as follows:

1. Tilde expansion (see Section 2.6.1 (on page 2239)), parameter expansion (see Section 2.6.2 (on page 2239)), command substitution (see Section 2.6.3 (on page 2242)), and arithmetic expansion (see Section 2.6.4 (on page 2243)) shall be performed, beginning to end. See item 5 in Section 2.3 (on page 2233).
2. Field splitting (see Section 2.6.5 (on page 2243)) shall be performed on the portions of the fields generated by step 1, unless *IFS* is null.
3. Pathname expansion (see Section 2.6.6 (on page 2244)) shall be performed, unless *set* **–f** is in effect.
4. Quote removal (see Section 2.6.7 (on page 2244)) shall always be performed last.

The expansions described in this section shall occur in the same shell environment as that in which the command is executed.

If the complete expansion appropriate for a word results in an empty field, that empty field shall be deleted from the list of fields that form the completely expanded command, unless the original word contained single-quote or double-quote characters.

The '$' character is used to introduce parameter expansion, command substitution, or arithmetic evaluation. If an unquoted '$' is followed by a character that is either not numeric, the name of one of the special parameters (see Section 2.5.2 (on page 2235)), a valid first character of a variable name, a left curly brace ('{') or a left parenthesis, the result is unspecified.

2.6.1 Tilde Expansion

A *tilde-prefix* consists of an unquoted tilde character at the beginning of a word, followed by all of the characters preceding the first unquoted slash in the word, or all the characters in the word if there is no slash. In an assignment (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 4.21, Variable Assignment), multiple tilde-prefixes can be used: at the beginning of the word (that is, following the equal sign of the assignment), following any unquoted colon, or both. A tilde-prefix in an assignment is terminated by the first unquoted colon or slash. If none of the characters in the tilde-prefix are quoted, the characters in the tilde-prefix following the tilde are treated as a possible login name from the user database. A portable login name cannot contain characters outside the set given in the description of the *LOGNAME* environment variable in the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.3, Other Environment Variables. If the login name is null (that is, the tilde-prefix contains only the tilde), the tilde-prefix is replaced by the value of the variable *HOME*. If *HOME* is unset, the results are unspecified. Otherwise, the tilde-prefix shall be replaced by a pathname of the initial working directory associated with the login name obtained using the *getpwnam*( ) function as defined in the System Interfaces volume of IEEE Std 1003.1-200x. If the system does not recognize the login name, the results are undefined.

2.6.2 Parameter Expansion

The format for parameter expansion is as follows:

```
${expression}
```

where *expression* consists of all characters until the matching '}'. Any '}' escaped by a backslash or within a quoted string, and characters in embedded arithmetic expansions, command substitutions, and variable expansions, shall not be examined in determining the matching '}'.

The simplest form for parameter expansion is:

```
${parameter}
```

The value, if any, of *parameter* shall be substituted.

The parameter name or symbol can be enclosed in braces, which are optional except for positional parameters with more than one digit or when *parameter* is followed by a character that could be interpreted as part of the name. The matching closing brace shall be determined by counting brace levels, skipping over enclosed quoted strings, and command substitutions.

If the parameter name or symbol is not enclosed in braces, the expansion shall use the longest valid name (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.230, Name), whether or not the symbol represented by that name exists.

If a parameter expansion occurs inside double-quotes:

- Pathname expansion shall not be performed on the results of the expansion.
- Field splitting shall not be performed on the results of the expansion, with the exception of '@'; see Section 2.5.2 (on page 2235).

In addition, a parameter expansion can be modified by using one of the following formats. In each case that a value of *word* is needed (based on the state of *parameter*, as described below), *word* shall be subjected to tilde expansion, parameter expansion, command substitution, and arithmetic expansion. If *word* is not needed, it shall not be expanded. The `'}'` character that delimits the following parameter expansion modifications shall be determined as described previously in this section and in Section 2.2.3 (on page 2232). (For example, ${**foo-bar**}**xyz**} would result in the expansion of **foo** followed by the string **xyz**} if **foo** is set, else the string `"barxyz}"`).

${*parameter*:–*word*} **Use Default Values**. If *parameter* is unset or null, the expansion of *word* shall be substituted; otherwise, the value of *parameter* shall be substituted.

${*parameter*:=*word*} **Assign Default Values**. If *parameter* is unset or null, the expansion of *word* shall be assigned to *parameter*. In all cases, the final value of *parameter* shall be substituted. Only variables, not positional parameters or special parameters, can be assigned in this way.

${*parameter*:?**[***word***]**} **Indicate Error if Null or Unset**. If *parameter* is unset or null, the expansion of *word* (or a message indicating it is unset if *word* is omitted) shall be written to standard error and the shell exits with a non-zero exit status. Otherwise, the value of *parameter* shall be substituted. An interactive shell need not exit.

${*parameter*:+*word*} **Use Alternative Value**. If *parameter* is unset or null, null shall be substituted; otherwise, the expansion of *word* shall be substituted.

In the parameter expansions shown previously, use of the colon in the format shall result in a test for a parameter that is unset or null; omission of the colon shall result in a test for a parameter that is only unset. The following table summarizes the effect of the colon:

	parameter **Set and Not Null**	*parameter* **Set But Null**	*parameter* **Unset**
${*parameter***:–***word***}**	substitute *parameter*	substitute *word*	substitute *word*
${*parameter***–***word***}**	substitute *parameter*	substitute null	substitute *word*
${*parameter***:=***word***}**	substitute *parameter*	assign *word*	assign *word*
${*parameter***=***word***}**	substitute *parameter*	substitute *parameter*	assign null
${*parameter***:?***word***}**	substitute *parameter*	error, exit	error, exit
${*parameter***?***word***}**	substitute *parameter*	substitute null	error, exit
${*parameter***:+***word***}**	substitute *word*	substitute null	substitute null
${*parameter***+***word***}**	substitute *word*	substitute *word*	substitute null

In all cases shown with ‘‘substitute’’, the expression is replaced with the value shown. In all cases shown with ‘‘assign’’, *parameter* is assigned that value, which also replaces the expression.

${*#parameter*} **String Length**. The length in characters of the value of *parameter* shall be substituted. If *parameter* is `'*'` or `'@'`, the result of the expansion is unspecified.

The following four varieties of parameter expansion provide for substring processing. In each case, pattern matching notation (see Section 2.13 (on page 2264)), rather than regular expression notation, shall be used to evaluate the patterns. If *parameter* is `'*'` or `'@'`, the result of the expansion is unspecified. Enclosing the full parameter expansion string in double-quotes shall not cause the following four varieties of pattern characters to be quoted, whereas quoting characters within the braces shall have this effect.

${*parameter%word*} **Remove Smallest Suffix Pattern**. The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the

smallest portion of the suffix matched by the *pattern* deleted.

${*parameter*%%*word*} **Remove Largest Suffix Pattern**. The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the largest portion of the suffix matched by the *pattern* deleted.

${*parameter*#*word*} **Remove Smallest Prefix Pattern**. The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the smallest portion of the prefix matched by the *pattern* deleted.

${*parameter*##*word*} **Remove Largest Prefix Pattern**. The *word* shall be expanded to produce a pattern. The parameter expansion shall then result in *parameter*, with the largest portion of the prefix matched by the *pattern* deleted.

Examples

${*parameter*:–*word*}

In this example, *ls* is executed only if *x* is null or unset. (The $(*ls*) command substitution notation is explained in Section 2.6.3 (on page 2242).)

```
${x:–$(ls)}
```

${*parameter*:=*word*}

```
unset X
echo ${X:=abc}
abc
```

${*parameter*:?*word*}

```
unset posix
echo ${posix:?}
sh: posix: parameter null or not set
```

${*parameter*:+*word*}

```
set a b c
echo ${3:+posix}
posix
```

${#*parameter*}

```
HOME=/usr/posix
echo ${#HOME}
10
```

${*parameter*%*word*}

```
x=file.c
echo ${x%.c}.o
file.o
```

${*parameter*%%*word*}

```
x=posix/src/std
echo ${x%%/*}
posix
```

${*parameter*#*word*}

```
x=$HOME/src/cmd
echo ${x#$HOME}
/src/cmd
```

${*parameter*##*word*}

```
x=/one/two/three
```

```
echo ${x##*/}
three
```

The double-quoting of patterns is different depending on where the double-quotes are placed:

`"${x#*}"` The asterisk is a pattern character.

`${x#"*"}` The literal asterisk is quoted and not special.

2.6.3 Command Substitution

Command substitution allows the output of a command to be substituted in place of the command name itself. Command substitution shall occur when the command is enclosed as follows:

`$(`*command*`)`

or (backquoted version):

`` `command` ``

The shell shall expand the command substitution by executing *command* in a subshell environment (see Section 2.12 (on page 2263)) and replacing the command substitution (the text of *command* plus the enclosing `"$()"` or backquotes) with the standard output of the command, removing sequences of one or more <newline>s at the end of the substitution. Embedded <newline>s before the end of the output shall not be removed; however, they may be treated as field delimiters and eliminated during field splitting, depending on the value of *IFS* and quoting that is in effect.

Within the backquoted style of command substitution, backslash shall retain its literal meaning, except when followed by: `'$'`, `` '`' ``, or `'\\'` (dollar sign, backquote, backslash). The search for the matching backquote shall be satisfied by the first backquote found without a preceding backslash; during this search, if a non-escaped backquote is encountered within a shell comment, a here-document, an embedded command substitution of the $(*command*) form, or a quoted string, undefined results occur. A single-quoted or double-quoted string that begins, but does not end, within the `` "`...`" `` sequence produces undefined results.

With the $(*command*) form, all characters following the open parenthesis to the matching closing parenthesis constitute the *command*. Any valid shell script can be used for *command*, except:

- A script consisting solely of redirections produces unspecified results
- See the restriction on single subshells described below

The results of command substitution shall not be processed for further tilde expansion, parameter expansion, command substitution, or arithmetic expansion. If a command substitution occurs inside double-quotes, it shall not be performed on the results of the substitution.

Command substitution can be nested. To specify nesting within the backquoted version, the application shall precede the inner backquotes with backslashes, for example:

`` \`command\` ``

If the command substitution consists of a single subshell, such as:

`$( (`*command*`) )`

a conforming application shall separate the `"$("` and `'('` into two tokens (that is, separate them with white space). This is required to avoid any ambiguities with arithmetic expansion.

2.6.4 Arithmetic Expansion

Arithmetic expansion provides a mechanism for evaluating an arithmetic expression and substituting its value. The format for arithmetic expansion shall be as follows:

```
$((expression))
```

The expression shall be treated as if it were in double-quotes, except that a double-quote inside the expression is not treated specially. The shell shall expand all tokens in the expression for parameter expansion, command substitution, and quote removal.

Next, the shell shall treat this as an arithmetic expression and substitute the value of the expression. The arithmetic expression shall be processed according to the rules given in Section 1.7.2.1 (on page 2207), with the following exceptions:

- Only signed long integer arithmetic is required.
- Only the decimal-constant, octal-constant, and hexadecimal-constant constants specified in the ISO C standard, Subclause 6.4.4.1 are required to be recognized as constants.
- The *sizeof*( ) operator and the prefix and postfix "++" and "−−" operators are not required.
- Selection, iteration, and jump statements are not supported.

As an extension, the shell may recognize arithmetic expressions beyond those listed. The shell may use a signed integer type with a rank larger than the rank of **signed long**. The shell may use a real-floating type instead of **signed long** as long as it does not affect the results in cases where there is no overflow. If the expression is invalid, the expansion fails and the shell shall write a message to standard error indicating the failure.

Examples

A simple example using arithmetic expansion:

```
# repeat a command 100 times
x=100
while [ $x –gt 0 ]
do
    command
    x=$(($x–1))
done
```

2.6.5 Field Splitting

After parameter expansion (Section 2.6.2 (on page 2239)), command substitution (Section 2.6.3 (on page 2242)), and arithmetic expansion (Section 2.6.4), the shell shall scan the results of expansions and substitutions that did not occur in double-quotes for field splitting and multiple fields can result.

The shell shall treat each character of the *IFS* as a delimiter and use the delimiters to split the results of parameter expansion and command substitution into fields.

1. If the value of *IFS* is a <space>, <tab>, and <newline>, or if it is unset, any sequence of <space>s, <tab>s, or <newline>s at the beginning or end of the input shall be ignored and any sequence of those characters within the input shall delimit a field. For example, the input:

   ```
   <newline><space><tab>foo<tab><tab>bar<space>
   ```

yields two fields, **foo** and **bar**.

2. If the value of *IFS* is null, no field splitting shall be performed.

3. Otherwise, the following rules shall be applied in sequence. The term ‘‘*IFS* white space’’ is used to mean any sequence (zero or more instances) of white space characters that are in the *IFS* value (for example, if *IFS* contains <space>/<comma>/<tab>, any sequence of <space>s and <tab>s is considered *IFS* white space).

   a. *IFS* white space shall be ignored at the beginning and end of the input.

   b. Each occurrence in the input of an *IFS* character that is not *IFS* white space, along with any adjacent *IFS* white space, shall delimit a field, as described previously.

   c. Non-zero-length *IFS* white space shall delimit a field.

2.6.6 Pathname Expansion

After field splitting, if *set* –**f** is not in effect, each field in the resulting command line shall be expanded using the algorithm described in Section 2.13 (on page 2264), qualified by the rules in Section 2.13.3 (on page 2265).

2.6.7 Quote Removal

The quote characters: `'\'`, `'\''`, and `'"'` (backslash, single-quote, double-quote) that were present in the original word shall be removed unless they have themselves been quoted.

2.7 Redirection

Redirection is used to open and close files for the current shell execution environment (see Section 2.12 (on page 2263)) or for any command. *Redirection operators* can be used with numbers representing file descriptors (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.165, File Descriptor) as described below.

The overall format used for redirection is:

```
[n]redir-op word
```

The number *n* is an optional decimal number designating the file descriptor number; the application shall ensure it is delimited from any preceding text and immediately precede the redirection operator *redir-op*. If *n* is quoted, the number shall not be recognized as part of the redirection expression. For example:

```
echo \2>a
```

writes the character 2 into file **a**. If any part of *redir-op* is quoted, no redirection expression is recognized. For example:

```
echo 2\>a
```

writes the characters 2>*a* to standard output. The optional number, redirection operator, and *word* shall not appear in the arguments provided to the command to be executed (if any).

Open files are represented by decimal numbers starting with zero. The largest possible value is implementation-defined; however, all implementations shall support at least 0 to 9, inclusive, for use by the application. These numbers are called *file descriptors*. The values 0, 1, and 2 have special meaning and conventional uses and are implied by certain redirection operations; they are referred to as *standard input*, *standard output*, and *standard error*, respectively. Programs usually take their input from standard input, and write output on standard output. Error

messages are usually written on standard error. The redirection operators can be preceded by one or more digits (with no intervening <blank>s allowed) to designate the file descriptor number.

If the redirection operator is "<<" or "<<–", the word that follows the redirection operator shall be subjected to quote removal; it is unspecified whether any of the other expansions occur. For the other redirection operators, the word that follows the redirection operator shall be subjected to tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote removal. Pathname expansion shall not be performed on the word by a non-interactive shell; an interactive shell may perform it, but shall do so only when the expansion would result in one word.

If more than one redirection operator is specified with a command, the order of evaluation is from beginning to end.

A failure to open or create a file shall cause a redirection to fail.

2.7.1 Redirecting Input

Input redirection shall cause the file whose name results from the expansion of *word* to be opened for reading on the designated file descriptor, or standard input if the file descriptor is not specified.

The general format for redirecting input is:

```
[n]<word
```

where the optional *n* represents the file descriptor number. If the number is omitted, the redirection shall refer to standard input (file descriptor 0).

2.7.2 Redirecting Output

The two general formats for redirecting output are:

```
[n]>word
[n]>|word
```

where the optional *n* represents the file descriptor number. If the number is omitted, the redirection shall refer to standard output (file descriptor 1).

Output redirection using the '>' format shall fail if the *noclobber* option is set (see the description of *set* –**C**) and the file named by the expansion of *word* exists and is a regular file. Otherwise, redirection using the '>' or ">|" formats shall cause the file whose name results from the expansion of *word* to be created and opened for output on the designated file descriptor, or standard output if none is specified. If the file does not exist, it shall be created; otherwise, it shall be truncated to be an empty file after being opened.

2.7.3 Appending Redirected Output

Appended output redirection shall cause the file whose name results from the expansion of word to be opened for output on the designated file descriptor. The file is opened as if the *open*( ) function as defined in the System Interfaces volume of IEEE Std 1003.1-200x was called with the O_APPEND flag. If the file does not exist, it shall be created.

The general format for appending redirected output is as follows:

```
[n]>>word
```

where the optional *n* represents the file descriptor number. If the number is omitted, the redirection refers to standard output (file descriptor 1).

2.7.4 Here-Document

The redirection operators "<<" and "<<–" both allow redirection of lines contained in a shell input file, known as a *here-document*, to the input of a command.

The here-document shall be treated as a single word that begins after the next <newline> and continues until there is a line containing only the delimiter and a <newline>, with no <blank>s in between. Then the next here-document starts, if there is one. The format is as follows:

```
[n]<<word
    here-document
delimiter
```

where the optional *n* represents the file descriptor number. If the number is omitted, the here-document refers to standard input (file descriptor 0).

If any character in *word* is quoted, the delimiter shall be formed by performing quote removal on *word*, and the here-document lines shall not be expanded. Otherwise, the delimiter shall be the *word* itself.

If no characters in *word* are quoted, all lines of the here-document shall be expanded for parameter expansion, command substitution, and arithmetic expansion. In this case, the backslash in the input behaves as the backslash inside double-quotes (see Section 2.2.3 (on page 2232)). However, the double-quote character (`'"'`) shall not be treated specially within a here-document, except when the double-quote appears within "`$()`", "``` ``` ```", or "`${}`".

If the redirection symbol is "<<–", all leading tab characters shall be stripped from input lines and the line containing the trailing delimiter. If more than one "<<" or "<<–" operator is specified on a line, the here-document associated with the first operator shall be supplied first by the application and shall be read first by the shell.

Examples

An example of a here-document follows:

```
cat <<eof1; cat <<eof2
Hi,
eof1
Helene.
eof2
```

2.7.5 Duplicating an Input File Descriptor

The redirection operator:

```
[n]<&word
```

shall duplicate one input file descriptor from another, or shall close one. If *word* evaluates to one or more digits, the file descriptor denoted by *n*, or standard input if *n* is not specified, shall be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent a file descriptor already open for input, a redirection error shall result; see Section 2.8.1 (on page 2247). If *word* evaluates to `'–'`, file descriptor *n*, or standard input if *n* is not specified, shall be closed. If *word* evaluates to something else, the behavior is unspecified.

2.7.6 Duplicating an Output File Descriptor

The redirection operator:

```
[n]>&word
```

shall duplicate one output file descriptor from another, or shall close one. If *word* evaluates to one or more digits, the file descriptor denoted by *n*, or standard output if *n* is not specified, shall be made to be a copy of the file descriptor denoted by *word*; if the digits in *word* do not represent a file descriptor already open for output, a redirection error shall result; see Section 2.8.1. If *word* evaluates to '-', file descriptor *n*, or standard output if *n* is not specified, is closed. If *word* evaluates to something else, the behavior is unspecified.

2.7.7 Open File Descriptors for Reading and Writing

The redirection operator:

```
[n]<>word
```

shall cause the file whose name is the expansion of *word* to be opened for both reading and writing on the file descriptor denoted by *n*, or standard input if *n* is not specified. If the file does not exist, it shall be created.

2.8 Exit Status and Errors

2.8.1 Consequences of Shell Errors

For a non-interactive shell, an error condition encountered by a special built-in (see Section 2.14 (on page 2266)) or other type of utility shall cause the shell to write a diagnostic message to standard error and exit as shown in the following table:

Error	Special Built-In	Other Utilities
Shell language syntax error	Shall exit	Shall exit
Utility syntax error (option or operand error)	Shall exit	Shall not exit
Redirection error	Shall exit	Shall not exit
Variable assignment error	Shall exit	Shall not exit
Expansion error	Shall exit	Shall exit
Command not found	N/A	May exit
Dot script not found	Shall exit	N/A

An expansion error is one that occurs when the shell expansions defined in Section 2.6 (on page 2238) are carried out (for example, "${x!y}", because '!' is not a valid operator); an implementation may treat these as syntax errors if it is able to detect them during tokenization, rather than during expansion.

If any of the errors shown as "shall exit" or "(may) exit" occur in a subshell, the subshell shall (respectively may) exit with a non-zero status, but the script containing the subshell shall not exit because of the error.

In all of the cases shown in the table, an interactive shell shall write a diagnostic message to standard error without exiting.

2.8.2 Exit Status for Commands

Each command has an exit status that can influence the behavior of other shell commands. The exit status of commands that are not utilities is documented in this section. The exit status of the standard utilities is documented in their respective sections.

If a command is not found, the exit status shall be 127. If the command name is found, but it is not an executable utility, the exit status shall be 126. Applications that invoke utilities without using the shell should use these exit status values to report similar errors.

If a command fails during word expansion or redirection, its exit status shall be greater than zero.

Internally, for purposes of deciding whether a command exits with a non-zero exit status, the shell shall recognize the entire status value retrieved for the command by the equivalent of the *wait*() function WEXITSTATUS macro (as defined in the System Interfaces volume of IEEE Std 1003.1-200x). When reporting the exit status with the special parameter `'?'`, the shell shall report the full eight bits of exit status available. The exit status of a command that terminated because it received a signal shall be reported as greater than 128.

2.9 Shell Commands

This section describes the basic structure of shell commands. The following command descriptions each describe a format of the command that is only used to aid the reader in recognizing the command type, and does not formally represent the syntax. Each description discusses the semantics of the command; for a formal definition of the command language, consult Section 2.10 (on page 2257).

A *command* is one of the following:

- *Simple command* (see Section 2.9.1)
- *Pipeline* (see Section 2.9.2 (on page 2250))
- *List* or *compound-list* (see Section 2.9.3 (on page 2251))
- *Compound command* (see Section 2.9.4 (on page 2253))
- *Function definition* (see Section 2.9.5 (on page 2256))

Unless otherwise stated, the exit status of a command shall be that of the last simple command executed by the command. There shall be no limit on the size of any shell command other than that imposed by the underlying system (memory constraints, {ARG_MAX}, and so on).

2.9.1 Simple Commands

A *simple command* is a sequence of optional variable assignments and redirections, in any sequence, optionally followed by words and redirections, terminated by a control operator.

When a given simple command is required to be executed (that is, when any conditional construct such as an AND-OR list or a **case** statement has not bypassed the simple command), the following expansions, assignments, and redirections shall all be performed from the beginning of the command text to the end:

1. The words that are recognized as variable assignments or redirections according to Section 2.10.2 (on page 2257) are saved for processing in steps 3 and 4.
2. The words that are not variable assignments or redirections shall be expanded. If any fields remain following their expansion, the first field shall be considered the command name

and remaining fields are the arguments for the command.

3. Redirections shall be performed as described in Section 2.7 (on page 2244).

4. Each variable assignment shall be expanded for tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote removal prior to assigning the value.

In the preceding list, the order of steps 3 and 4 may be reversed for the processing of special built-in utilities; see Section 2.14 (on page 2266).

If no command name results, variable assignments shall affect the current execution environment. Otherwise, the variable assignments shall be exported for the execution environment of the command and shall not affect the current execution environment (except for special built-ins). If any of the variable assignments attempt to assign a value to a read-only variable, a variable assignment error shall occur. See Section 2.8.1 (on page 2247) for the consequences of these errors.

If there is no command name, any redirections shall be performed in a subshell environment; it is unspecified whether this subshell environment is the same one as that used for a command substitution within the command. (To affect the current execution environment, see the *exec* (on page 2277) special built-in.) If any of the redirections performed in the current shell execution environment fail, the command shall immediately fail with an exit status greater than zero, and the shell shall write an error message indicating the failure. See Section 2.8.1 (on page 2247) for the consequences of these failures on interactive and non-interactive shells.

If there is a command name, execution shall continue as described in Section 2.9.1.1. If there is no command name, but the command contained a command substitution, the command shall complete with the exit status of the last command substitution performed. Otherwise, the command shall complete with a zero exit status.

2.9.1.1 *Command Search and Execution*

If a simple command results in a command name and an optional list of arguments, the following actions shall be performed:

1. If the command name does not contain any slashes, the first successful step in the following sequence shall occur:

   a. If the command name matches the name of a special built-in utility, that special built-in utility shall be invoked.

   b. If the command name matches the name of a function known to this shell, the function shall be invoked as described in Section 2.9.5 (on page 2256). If the implementation has provided a standard utility in the form of a function, it shall not be recognized at this point. It shall be invoked in conjunction with the path search in step 1d.

   c. If the command name matches the name of a utility listed in the following table, that utility shall be invoked.

alias	*false*	*jobs*	*true*
bg	*fc*	*kill*	*umask*
cd	*fg*	*newgrp*	*unalias*
command	*getopts*	*read*	*wait*

   d. Otherwise, the command shall be searched for using the *PATH* environment variable as described in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables:

i. If the search is successful:

   a. If the system has implemented the utility as a regular built-in or as a shell function, it shall be invoked at this point in the path search.

   b. Otherwise, the shell executes the utility in a separate utility environment (see Section 2.12 (on page 2263)) with actions equivalent to calling the *execve*( ) function as defined in the System Interfaces volume of IEEE Std 1003.1-200x with the *path* argument set to the pathname resulting from the search, *arg*0 set to the command name, and the remaining arguments set to the operands, if any.

      If the *execve*( ) function fails due to an error equivalent to the [ENOEXEC] error defined in the System Interfaces volume of IEEE Std 1003.1-200x, the shell shall execute a command equivalent to having a shell invoked with the command name as its first operand, with any remaining arguments passed to the new shell. If the executable file is not a text file, the shell may bypass this command execution. In this case, it shall write an error message, and shall return an exit status of 126.

   Once a utility has been searched for and found (either as a result of this specific search or as part of an unspecified shell start-up activity), an implementation may remember its location and need not search for the utility again unless the *PATH* variable has been the subject of an assignment. If the remembered location fails for a subsequent invocation, the shell shall repeat the search to find the new location for the utility, if any.

ii. If the search is unsuccessful, the command shall fail with an exit status of 127 and the shell shall write an error message.

2. If the command name contains at least one slash, the shell shall execute the utility in a separate utility environment with actions equivalent to calling the *execve*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x with the *path* and *arg*0 arguments set to the command name, and the remaining arguments set to the operands, if any.

   If the *execve*( ) function fails due to an error equivalent to the [ENOEXEC] error, the shell shall execute a command equivalent to having a shell invoked with the command name as its first operand, with any remaining arguments passed to the new shell. If the executable file is not a text file, the shell may bypass this command execution. In this case, it shall write an error message and shall return an exit status of 126.

2.9.2 Pipelines

A *pipeline* is a sequence of one or more commands separated by the control operator '|'. The standard output of all but the last command shall be connected to the standard input of the next command.

The format for a pipeline is:

```
[!] command1 [ | command2 ...]
```

The standard output of *command1* shall be connected to the standard input of *command2*. The standard input, standard output, or both of a command shall be considered to be assigned by the pipeline before any redirection specified by redirection operators that are part of the command (see Section 2.7 (on page 2244)).

If the pipeline is not in the background (see Section 2.9.3.1 (on page 2252)), the shell shall wait for the last command specified in the pipeline to complete, and may also wait for all commands to complete.

Exit Status

If the reserved word `!` does not precede the pipeline, the exit status shall be the exit status of the last command specified in the pipeline. Otherwise, the exit status shall be the logical NOT of the exit status of the last command. That is, if the last command returns zero, the exit status shall be 1; if the last command returns greater than zero, the exit status shall be zero.

2.9.3 Lists

An *AND-OR list* is a sequence of one or more pipelines separated by the operators "&&" and "||".

A *list* is a sequence of one or more AND-OR lists separated by the operators `';'` and `'&'` and optionally terminated by `';'`, `'&'`, or <newline>.

The operators "&&" and "||" shall have equal precedence and shall be evaluated with left associativity. For example, both of the following commands write solely **bar** to standard output:

```
false && echo foo || echo bar
true || echo foo && echo bar
```

A `';'` or <newline> terminator shall cause the preceding AND-OR list to be executed sequentially; an `'&'` shall cause asynchronous execution of the preceding AND-OR list.

The term *compound-list* is derived from the grammar in Section 2.10 (on page 2257); it is equivalent to a sequence of *lists*, separated by <newline>s, that can be preceded or followed by an arbitrary number of <newline>s.

Examples

The following is an example that illustrates <newline>s in compound-lists:

```
while
    # a couple of <newline>s

    # a list
    date && who || ls; cat file
    # a couple of <newline>s

    # another list
    wc file > output & true
do
    # 2 lists
    ls
    cat file
done
```

2058 *2.9.3.1 Asynchronous Lists*

2059 If a command is terminated by the control operator ampersand ('&'), the shell shall execute the
2060 command asynchronously in a subshell. This means that the shell shall not wait for the
2061 command to finish before executing the next command.

2062 The format for running a command in the background is:

2063 *command1* & [*command2* & ...] |

2064 The standard input for an asynchronous list, before any explicit redirections are performed, shall
2065 be considered to be assigned to a file that has the same properties as **/dev/null**. If it is an
2066 interactive shell, this need not happen. In all cases, explicit redirection of standard input shall
2067 override this activity.

2068 When an element of an asynchronous list (the portion of the list ended by an ampersand, such as
2069 *command1*, above) is started by the shell, the process ID of the last command in the asynchronous
2070 list element shall become known in the current shell execution environment; see Section 2.12 (on
2071 page 2263). This process ID shall remain known until:

2072 1. The command terminates and the application waits for the process ID.

2073 2. Another asynchronous list invoked before "$!" (corresponding to the previous
2074 asynchronous list) is expanded in the current execution environment.

2075 The implementation need not retain more than the {CHILD_MAX} most recent entries in its list
2076 of known process IDs in the current shell execution environment.

2077 **Exit Status**

2078 The exit status of an asynchronous list shall be zero.

2079 *2.9.3.2 Sequential Lists*

2080 Commands that are separated by a semicolon (';') shall be executed sequentially.

2081 The format for executing commands sequentially shall be:

2082 *command1* [; *command2*] ... |

2083 Each command shall be expanded and executed in the order specified.

2084 **Exit Status**

2085 The exit status of a sequential list shall be the exit status of the last command in the list.

2086 *2.9.3.3 AND Lists*

2087 The control operator "&&" denotes an AND list. The format shall be:

2088 *command1* [&& *command2*] ... |

2089 First *command1* shall be executed. If its exit status is zero, *command2* shall be executed, and so on,
2090 until a command has a non-zero exit status or there are no more commands left to execute. The
2091 commands are expanded only if they are executed.

Exit Status

The exit status of an AND list shall be the exit status of the last command that is executed in the list.

2.9.3.4 OR Lists

The control operator "||" denotes an OR List. The format shall be:

```
command1 [ || command2] ...
```

First, *command1* shall be executed. If its exit status is non-zero, *command2* shall be executed, and so on, until a command has a zero exit status or there are no more commands left to execute.

Exit Status

The exit status of an OR list shall be the exit status of the last command that is executed in the list.

2.9.4 Compound Commands

The shell has several programming constructs that are *compound commands*, which provide control flow for commands. Each of these compound commands has a reserved word or control operator at the beginning, and a corresponding terminator reserved word or operator at the end. In addition, each can be followed by redirections on the same line as the terminator. Each redirection shall apply to all the commands within the compound command that do not explicitly override that redirection.

2.9.4.1 Grouping Commands

The format for grouping commands is as follows:

(*compound-list*) Execute *compound-list* in a subshell environment; see Section 2.12 (on page 2263). Variable assignments and built-in commands that affect the environment shall not remain in effect after the list finishes.

{ *compound-list*;} Execute *compound-list* in the current process environment. The semicolon shown here is an example of a control operator delimiting the **}** reserved word. Other delimiters are possible, as shown in Section 2.10 (on page 2257); a <newline> is frequently used.

Exit Status

The exit status of a grouping command shall be the exit status of *list*.

2.9.4.2 For Loop

The **for** loop shall execute a sequence of commands for each member in a list of *items*. The **for** loop requires that the reserved words **do** and **done** be used to delimit the sequence of commands.

The format for the **for** loop is as follows:

```
for name [ in [word ... ]]
do
    compound-list
done
```

First, the list of words following **in** shall be expanded to generate a list of items. Then, the variable *name* shall be set to each item, in turn, and the *compound-list* executed each time. If no items result from the expansion, the *compound-list* shall not be executed. Omitting:

```
in word ...
```

shall be equivalent to:

```
in "$@"
```

Exit Status

The exit status of a **for** command shall be the exit status of the last command that executes. If there are no items, the exit status shall be zero.

2.9.4.3 *Case Conditional Construct*

The conditional construct **case** shall execute the *compound-list* corresponding to the first one of several *patterns* (see Section 2.13 (on page 2264)) that is matched by the string resulting from the tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote removal of the given word. The reserved word **in** shall denote the beginning of the patterns to be matched. Multiple patterns with the same *compound-list* shall be delimited by the '|' symbol. The control operator ')' terminates a list of patterns corresponding to a given action. The *compound-list* for each list of patterns, with the possible exception of the last, shall be terminated with ";;". The **case** construct terminates with the reserved word **esac** (**case** reversed).

The format for the **case** construct is as follows:

```
case word in
    [(]pattern1) compound-list;;
    [[(]pattern[ | pattern] ... ) compound-list;;] ...
    [[(]pattern[ | pattern] ... ) compound-list]
esac
```

The ";;" is optional for the last *compound-list*.

In order from the beginning to the end of the **case** statement, each *pattern* that labels a *compound-list* shall be subjected to tilde expansion, parameter expansion, command substitution, and arithmetic expansion, and the result of these expansions shall be compared against the expansion of *word*, according to the rules described in Section 2.13 (on page 2264) (which also describes the effect of quoting parts of the pattern). After the first match, no more patterns shall be expanded, and the *compound-list* shall be executed. The order of expansion and comparison of multiple *pattern*s that label a *compound-list* statement is unspecified.

Exit Status

The exit status of **case** shall be zero if no patterns are matched. Otherwise, the exit status shall be the exit status of the last command executed in the *compound-list*.

2.9.4.4 *If Conditional Construct*

The **if** command shall execute a *compound-list* and use its exit status to determine whether to execute another *compound-list*.

The format for the **if** construct is as follows:

```
if compound-list
then
    compound-list
[elif compound-list
then
    compound-list] ...
[else
    compound-list]
```

The **if** *compound-list* shall be executed; if its exit status is zero, the **then** *compound-list* shall be executed and the command shall complete. Otherwise, each **elif** *compound-list* shall be executed, in turn, and if its exit status is zero, the **then** *compound-list* shall be executed and the command shall complete. Otherwise, the **else** *compound-list* shall be executed.

Exit Status

The exit status of the **if** command shall be the exit status of the **then** or **else** *compound-list* that was executed, or zero, if none was executed.

2.9.4.5 *While Loop*

The **while** loop shall continuously execute one *compound-list* as long as another *compound-list* has a zero exit status.

The format of the **while** loop is as follows:

```
while compound-list-1
do
    compound-list-2
done
```

The *compound-list-1* shall be executed, and if it has a non-zero exit status, the **while** command shall complete. Otherwise, the *compound-list-2* shall be executed, and the process shall repeat.

Exit Status

The exit status of the **while** loop shall be the exit status of the last *compound-list-2* executed, or zero if none was executed.

2.9.4.6 *Until Loop*

The **until** loop shall continuously execute one *compound-list* as long as another *compound-list* has a non-zero exit status.

The format of the **until** loop is as follows:

```
until compound-list-1
do
    compound-list-2
done
```

The *compound-list-1* shall be executed, and if it has a zero exit status, the **until** command completes. Otherwise, the *compound-list-2* shall be executed, and the process repeats.

Exit Status

The exit status of the **until** loop shall be the exit status of the last *compound-list-2* executed, or zero if none was executed.

2.9.5 Function Definition Command

A function is a user-defined name that is used as a simple command to call a compound command with new positional parameters. A function is defined with a *function definition command*.

The format of a function definition command is as follows:

```
fname() compound-command[io-redirect ...]
```

The function is named *fname*; the application shall ensure that it is a name (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.230, Name). An implementation may allow other characters in a function name as an extension. The implementation shall maintain separate name spaces for functions and variables.

The argument *compound-command* represents a compound command, as described in Section 2.9.4 (on page 2253).

When the function is declared, none of the expansions in Section 2.6 (on page 2238) shall be performed on the text in *compound-command* or *io-redirect*; all expansions shall be performed as normal each time the function is called. Similarly, the optional *io-redirect* redirections and any variable assignments within *compound-command* shall be performed during the execution of the function itself, not the function definition. See Section 2.8.1 (on page 2247) for the consequences of failures of these operations on interactive and non-interactive shells.

When a function is executed, it shall have the syntax-error and variable-assignment properties described for special built-in utilities in the enumerated list at the beginning of Section 2.14 (on page 2266).

The *compound-command* shall be executed whenever the function name is specified as the name of a simple command (see Section 2.9.1.1 (on page 2249)). The operands to the command temporarily shall become the positional parameters during the execution of the *compound-command*; the special parameter `'#'` also shall be changed to reflect the number of operands. The special parameter 0 shall be unchanged. When the function completes, the values of the positional parameters and the special parameter `'#'` shall be restored to the values they had before the function was executed. If the special built-in *return* is executed in the *compound-command*, the function completes and execution shall resume with the next command after the function call.

Exit Status

The exit status of a function definition shall be zero if the function was declared successfully; otherwise, it shall be greater than zero. The exit status of a function invocation shall be the exit status of the last command executed by the function.

2.10 Shell Grammar

The following grammar defines the Shell Command Language. This formal syntax shall take precedence over the preceding text syntax description.

2.10.1 Shell Grammar Lexical Conventions

The input language to the shell must be first recognized at the character level. The resulting tokens shall be classified by their immediate context according to the following rules (applied in order). These rules shall be used to determine what a ''token'' is that is subject to parsing at the token level. The rules for token recognition in Section 2.3 (on page 2233) shall apply.

1. A <newline> shall be returned as the token identifier **NEWLINE**.
2. If the token is an operator, the token identifier for that operator shall result.
3. If the string consists solely of digits and the delimiter character is one of `'<'` or `'>'`, the token identifier **IO_NUMBER** shall be returned.
4. Otherwise, the token identifier **TOKEN** results.

Further distinction on **TOKEN** is context-dependent. It may be that the same **TOKEN** yields **WORD**, a **NAME**, an **ASSIGNMENT**, or one of the reserved words below, dependent upon the context. Some of the productions in the grammar below are annotated with a rule number from the following list. When a **TOKEN** is seen where one of those annotated productions could be used to reduce the symbol, the applicable rule shall be applied to convert the token identifier type of the **TOKEN** to a token identifier acceptable at that point in the grammar. The reduction shall then proceed based upon the token identifier type yielded by the rule applied. When more than one rule applies, the highest numbered rule shall apply (which in turn may refer to another rule). (Note that except in rule 7, the presence of an `'='` in the token has no effect.)

The **WORD** tokens shall have the word expansion rules applied to them immediately before the associated command is executed, not at the time the command is parsed.

2.10.2 Shell Grammar Rules

1. [Command Name]

   When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word shall result. Otherwise, the token **WORD** shall be returned. Also, if the parser is in any state where only a reserved word could be the next correct token, proceed as above.

   Note: Because at this point quote marks are retained in the token, quoted strings cannot be recognized as reserved words. This rule also implies that reserved words are not recognized except in certain positions in the input, such as after a <newline> or semicolon; the grammar presumes that if the reserved word is intended, it is properly delimited by the user, and does not attempt to reflect that requirement directly. Also note that line joining is done before tokenization, as described in Section 2.2.1 (on page 2232), so escaped <newline>s are already removed at this point.

   Rule 1 is not directly referenced in the grammar, but is referred to by other rules, or applies globally.

2. [Redirection to or from filename]

   The expansions specified in Section 2.7 (on page 2244) shall occur. As specified there, exactly one field can result (or the result is unspecified), and there are additional requirements on pathname expansion.

3. [Redirection from here-document]

   Quote removal shall be applied to the word to determine the delimiter that is used to find the end of the here-document that begins after the next <newline>.

4. [Case statement termination]

   When the **TOKEN** is exactly the reserved word **esac**, the token identifier for **esac** shall result. Otherwise, the token **WORD** shall be returned.

5. [**NAME** in **for**]

   When the **TOKEN** meets the requirements for a name (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.230, Name), the token identifier **NAME** shall result. Otherwise, the token **WORD** shall be returned.

6. [Third word of **for** and **case**]

   When the **TOKEN** is exactly the reserved word **in**, the token identifier for **in** shall result. Otherwise, the token **WORD** shall be returned. (As indicated in the grammar, a *linebreak* precedes the token **in**. If <newline>s are present at the indicated location, it is the token after them that is treated in this fashion.)

7. [Assignment preceding command name]

   a. [When the first word]

      If the **TOKEN** does not contain the character `'='`, rule 1 is applied. Otherwise, 7b shall be applied.

   b. [Not the first word]

      If the **TOKEN** contains the equal sign character:

      — If it begins with `'='`, the token **WORD** shall be returned.

      — If all the characters preceding `'='` form a valid name (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.230, Name), the token **ASSIGNMENT_WORD** shall be returned. (Quoted characters cannot participate in forming a valid name.)

      — Otherwise, it is unspecified whether it is **ASSIGNMENT_WORD** or **WORD** that is returned.

   Assignment to the **NAME** shall occur as specified in Section 2.9.1 (on page 2248).

8. [**NAME** in function]

   When the **TOKEN** is exactly a reserved word, the token identifier for that reserved word shall result. Otherwise, when the **TOKEN** meets the requirements for a name, the token identifier **NAME** shall result. Otherwise, rule 7 applies.

9. [Body of function]

   Word expansion and assignment shall never occur, even when required by the rules above, when this rule is being parsed. Each **TOKEN** that might either be expanded or have assignment applied to it shall instead be returned as a single **WORD** consisting only of characters that are exactly the token described in Section 2.3 (on page 2233).

```
2324        /* ------------------------------------------------------
2325           The grammar symbols
2326           ------------------------------------------------------ */
2327        %token  WORD
2328        %token  ASSIGNMENT_WORD
2329        %token  NAME
2330        %token  NEWLINE
2331        %token  IO_NUMBER
2332        /* The following are the operators mentioned above. */
2333        %token  AND_IF    OR_IF    DSEMI
2334        /*      '&&'      '||'     ';;'    */
2335        %token  DLESS  DGREAT  LESSAND  GREATAND  LESSGREAT  DLESSDASH
2336        /*      '<<'   '>>'    '<&'     '>&'      '<>'       '<<-'   */
2337        %token  CLOBBER
2338        /*      '>|'   */
2339        /* The following are the reserved words. */
2340        %token  If    Then    Else    Elif    Fi    Do    Done
2341        /*      'if'  'then'  'else'  'elif'  'fi'  'do'  'done'   */
2342        %token  Case    Esac    While    Until    For
2343        /*      'case'  'esac'  'while'  'until'  'for'   */
2344        /* These are reserved words, not operator tokens, and are
2345           recognized when reserved words are recognized. */
2346        %token  Lbrace    Rbrace    Bang
2347        /*      '{'       '}'       '!'   */
2348        %token  In
2349        /*      'in'   */
2350        /* ------------------------------------------------------
2351           The Grammar
2352           ------------------------------------------------------ */
2353        %start  complete_command
2354        %%
2355        complete_command : list separator
2356                         | list
2357                         ;
2358        list             : list separator_op and_or
2359                         |                   and_or
2360                         ;
2361        and_or           :                         pipeline
2362                         | and_or AND_IF linebreak pipeline
2363                         | and_or OR_IF  linebreak pipeline
2364                         ;
2365        pipeline         :      pipe_sequence
2366                         | Bang pipe_sequence
2367                         ;
2368        pipe_sequence    :                             command
2369                         | pipe_sequence '|' linebreak command
```

```
                        ;
command                 : simple_command
                        | compound_command
                        | compound_command redirect_list
                        | function_definition
                        ;
compound_command        : brace_group
                        | subshell
                        | for_clause
                        | case_clause
                        | if_clause
                        | while_clause
                        | until_clause
                        ;
subshell                : '(' compound_list ')'
                        ;
compound_list           :              term
                        | newline_list term
                        |              term separator
                        | newline_list term separator
                        ;
term                    : term separator and_or
                        |                and_or
                        ;
for_clause              : For name linebreak                            do_group
                        | For name linebreak in          sequential_sep do_group
                        | For name linebreak in wordlist sequential_sep do_group
                        ;
name                    : NAME                     /* Apply rule 5 */
                        ;
in                      : In                       /* Apply rule 6 */
                        ;
wordlist                : wordlist WORD
                        |          WORD
                        ;
case_clause             : Case WORD linebreak in linebreak case_list Esac
                        | Case WORD linebreak in linebreak case_list_ns Esac
                        | Case WORD linebreak in linebreak Esac
                        ;
case_list_ns            : case_list case_item_ns
                        | case_item_ns
                        ;
case_list               : case_list case_item
                        | case_item
                        ;
case_item_ns            : pattern ')' linebreak linebreak
                        | pattern ')' compound_list linebreak
                        | '(' pattern ')' linebreak linebreak
                        | '(' pattern ')' compound_list linebreak
                        ;
case_item               : pattern ')' linebreak DSEMI linebreak
                        | pattern ')' compound_list linebreak
```

```
                        | '(' pattern ')' linebreak linebreak
                        | '(' pattern ')' compound_list linebreak
                        ;
pattern                 :             WORD         /* Apply rule 4 */
                        | pattern '|' WORD         /* Do not apply rule (4) */
                        ;
if_clause               : If compound_list Then compound_list else_part Fi
                        | If compound_list Then compound_list           Fi
                        ;
else_part               : Elif compound_list Then else_part
                        | Else compound_list
                        ;
while_clause            : While compound_list do_group
                        ;
until_clause            : Until compound_list do_group
                        ;
function_definition : fname '(' ')' linebreak function_body
                        ;
function_body           : compound_command                /* Apply rule 9 */
                        | compound_command redirect_list  /* Apply rule 9 */
                        ;
fname                   : NAME                            /* Apply rule 8 */
                        ;
brace_group             : Lbrace compound_list Rbrace
                        ;
do_group                : Do compound_list Done
                        ;
simple_command          : cmd_prefix cmd_word cmd_suffix
                        | cmd_prefix cmd_word
                        | cmd_prefix
                        | cmd_name cmd_suffix
                        | cmd_name
                        ;
cmd_name                : WORD                   /* Apply rule 7a */
                        ;
cmd_word                : WORD                   /* Apply rule 7b */
                        ;
cmd_prefix              :            io_redirect
                        | cmd_prefix io_redirect
                        |            ASSIGNMENT_WORD
                        | cmd_prefix ASSIGNMENT_WORD
                        ;
cmd_suffix              :            io_redirect
                        | cmd_suffix io_redirect
                        |            WORD
                        | cmd_suffix WORD
                        ;
redirect_list           :               io_redirect
                        | redirect_list io_redirect
                        ;
io_redirect             :           io_file
                        | IO_NUMBER io_file
```

```
                    |           io_here
                    | IO_NUMBER io_here
                    ;
io_file             : '<'       filename
                    | LESSAND   filename
                    | '>'       filename
                    | GREATAND  filename
                    | DGREAT    filename
                    | LESSGREAT filename
                    | CLOBBER   filename
                    ;
filename            : WORD                      /* Apply rule 2 */
                    ;
io_here             : DLESS     here_end
                    | DLESSDASH here_end
                    ;
here_end            : WORD                      /* Apply rule 3 */
                    ;
newline_list        :              NEWLINE
                    | newline_list NEWLINE
                    ;
linebreak           : newline_list
                    | /* empty */
                    ;
separator_op        : '&'
                    | ';'
                    ;
separator           : separator_op linebreak
                    | newline_list
                    ;
sequential_sep      : ';' linebreak
                    | newline_list
                    ;
```

2.11 Signals and Error Handling

When a command is in an asynchronous list, the shell shall prevent SIGQUIT and SIGINT signals from the keyboard from interrupting the command. Otherwise, signals shall have the values inherited by the shell from its parent (see also the *trap* (on page 2297) special built-in).

When a signal for which a trap has been set is received while the shell is waiting for the completion of a utility executing a foreground command, the trap associated with that signal shall not be executed until after the foreground command has completed. When the shell is waiting, by means of the *wait* utility, for asynchronous commands to complete, the reception of a signal for which a trap has been set shall cause the *wait* utility to return immediately with an exit status >128, immediately after which the trap associated with that signal shall be taken.

If multiple signals are pending for the shell for which there are associated trap actions, the order of execution of trap actions is unspecified.

2.12 Shell Execution Environment

A shell execution environment consists of the following:

- Open files inherited upon invocation of the shell, plus open files controlled by *exec*
- Working directory as set by *cd*
- File creation mask set by *umask*
- Current traps set by *trap*
- Shell parameters that are set by variable assignment (see the *set* (on page 2287) special built-in) or from the System Interfaces volume of IEEE Std 1003.1-200x environment inherited by the shell when it begins (see the *export* (on page 2281) special built-in)
- Shell functions; see Section 2.9.5 (on page 2256)
- Options turned on at invocation or by *set*
- Process IDs of the last commands in asynchronous lists known to this shell environment; see Section 2.9.3.1 (on page 2252)
- Shell aliases; see Section 2.3.1 (on page 2234)

Utilities other than the special built-ins (see Section 2.14 (on page 2266)) shall be invoked in a separate environment that consists of the following. The initial value of these objects shall be the same as that for the parent shell, except as noted below.

- Open files inherited on invocation of the shell, open files controlled by the *exec* special built-in plus any modifications, and additions specified by any redirections to the utility
- Current working directory
- File creation mask
- If the utility is a shell script, traps caught by the shell shall be set to the default values and traps ignored by the shell shall be set to be ignored by the utility; if the utility is not a shell script, the trap actions (default or ignore) shall be mapped into the appropriate signal handling actions for the utility
- Variables with the *export* attribute, along with those explicitly exported for the duration of the command, shall be passed to the utility as System Interfaces volume of IEEE Std 1003.1-200x environment variables

The environment of the shell process shall not be changed by the utility unless explicitly specified by the utility description (for example, *cd* and *umask*).

A subshell environment shall be created as a duplicate of the shell environment, except that signal traps set by that shell environment shall be set to the default values. Changes made to the subshell environment shall not affect the shell environment. Command substitution, commands that are grouped with parentheses, and asynchronous lists shall be executed in a subshell environment. Additionally, each command of a multi-command pipeline is in a subshell environment; as an extension, however, any or all commands in a pipeline may be executed in the current environment. All other commands shall be executed in the current shell environment.

2.13 Pattern Matching Notation

The pattern matching notation described in this section is used to specify patterns for matching strings in the shell. Historically, pattern matching notation is related to, but slightly different from, the regular expression notation described in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 9, Regular Expressions. For this reason, the description of the rules for this pattern matching notation are based on the description of regular expression notation, modified to include backslash escape processing.

2.13.1 Patterns Matching a Single Character

The following *patterns matching a single character* shall match a single character: *ordinary characters*, *special pattern characters*, and *pattern bracket expressions*. The pattern bracket expression also shall match a single collating element. A backslash character shall escape the following character. The escaping backslash shall be discarded.

An ordinary character is a pattern that shall match itself. It can be any character in the supported character set except for NUL, those special shell characters in Section 2.2 (on page 2232) that require quoting, and the following three special pattern characters. Matching shall be based on the bit pattern used for encoding the character, not on the graphic representation of the character. If any character (ordinary, shell special, or pattern special) is quoted, that pattern shall match the character itself. The shell special characters always require quoting.

When unquoted and outside a bracket expression, the following three characters shall have special meaning in the specification of patterns:

? A question-mark is a pattern that shall match any character.

* An asterisk is a pattern that shall match multiple characters, as described in Section 2.13.2.

[The open bracket shall introduce a pattern bracket expression.

The description of basic regular expression bracket expressions in the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.3.5, RE Bracket Expression shall also apply to the pattern bracket expression, except that the exclamation mark character (`'!'`) shall replace the circumflex character (`'^'`) in its role in a *non-matching list* in the regular expression notation. A bracket expression starting with an unquoted circumflex character produces unspecified results.

When pattern matching is used where shell quote removal is not performed (such as in the argument to the *find name* primary when *find* is being called using one of the *exec* functions as defined in the System Interfaces volume of IEEE Std 1003.1-200x, or in the *pattern* argument to the *fnmatch*( ) function), special characters can be escaped to remove their special meaning by preceding them with a backslash character. This escaping backslash is discarded. The sequence "\\" represents one literal backslash. All of the requirements and effects of quoting on ordinary, shell special, and special pattern characters shall apply to escaping in this context.

2.13.2 Patterns Matching Multiple Characters

The following rules are used to construct *patterns matching multiple characters* from *patterns matching a single character*:

1. The asterisk (`'*'`) is a pattern that shall match any string, including the null string.
2. The concatenation of *patterns matching a single character* is a valid pattern that shall match the concatenation of the single characters or collating elements matched by each of the concatenated patterns.

3. The concatenation of one or more *patterns matching a single character* with one or more asterisks is a valid pattern. In such patterns, each asterisk shall match a string of zero or more characters, matching the greatest possible number of characters that still allows the remainder of the pattern to match the string.

2.13.3 Patterns Used for Filename Expansion

The rules described so far in Section 2.13.1 (on page 2264) and Section 2.13.2 (on page 2264) are qualified by the following rules that apply when pattern matching notation is used for filename expansion:

1. The slash character in a pathname shall be explicitly matched by using one or more slashes in the pattern; it shall neither be matched by the asterisk or question-mark special characters nor by a bracket expression. Slashes in the pattern shall be identified before bracket expressions; thus, a slash cannot be included in a pattern bracket expression used for filename expansion. If a slash character is found following an unescaped open square bracket character before a corresponding closing square bracket is found, the open bracket shall be treated as an ordinary character. For example, the pattern `"a[b/c]d"` does not match such pathnames as **abd** or **a/d**. It only matches a pathname of literally **a[b/c]d**.

2. If a filename begins with a period (`'.'`), the period shall be explicitly matched by using a period as the first character of the pattern or immediately following a slash character. The leading period shall not be matched by:

   - The asterisk or question-mark special characters
   - A bracket expression containing a non-matching list, such as `"[!a]"`, a range expression, such as `"[%–0]"`, or a character class expression, such as `"[[:punct:]]"`

   It is unspecified whether an explicit period in a bracket expression matching list, such as `"[.abc]"`, can match a leading period in a filename.

3. Specified patterns shall be matched against existing filenames and pathnames, as appropriate. Each component that contains a pattern character shall require read permission in the directory containing that component. Any component, except the last, that does not contain a pattern character shall require search permission. For example, given the pattern:

   ```
   /foo/bar/x*/bam
   ```

   search permission is needed for directories **/** and **foo**, search and read permissions are needed for directory **bar**, and search permission is needed for each **x*** directory. If the pattern matches any existing filenames or pathnames, the pattern shall be replaced with those filenames and pathnames, sorted according to the collating sequence in effect in the current locale. If the pattern contains an invalid bracket expression or does not match any existing filenames or pathnames, the pattern string shall be left unchanged.

2.14 Special Built-In Utilities

The following *special built-in* utilities shall be supported in the shell command language. The output of each command, if any, shall be written to standard output, subject to the normal redirection and piping possible with all commands.

The term *built-in* implies that the shell can execute the utility directly and does not need to search for it. An implementation may choose to make any utility a built-in; however, the special built-in utilities described here differ from regular built-in utilities in two respects:

1. A syntax error in a special built-in utility may cause a shell executing that utility to abort, while a syntax error in a regular built-in utility shall not cause a shell executing that utility to abort. (See Section 2.8.1 (on page 2247) for the consequences of errors on interactive and non-interactive shells.) If a special built-in utility encountering a syntax error does not abort the shell, its exit value shall be non-zero.
2. Variable assignments specified with special built-in utilities remain in effect after the built-in completes; this shall not be the case with a regular built-in or other utility.

The special built-in utilities in this section need not be provided in a manner accessible via the *exec* family of functions defined in the System Interfaces volume of IEEE Std 1003.1-200x.

Some of the special built-ins are described as conforming to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines. For those that are not, the requirement in Section 1.11 (on page 2221) that "−−" be recognized as a first argument to be discarded does not apply and a conforming application shall not use that argument.

NAME

break — exit from for, while, or until loop

SYNOPSIS

```
break [n]
```

DESCRIPTION

The *break* utility shall exit from the smallest enclosing **for**, **while**, or **until** loop, if any; or from the *n*th enclosing loop if *n* is specified. The value of *n* is an unsigned decimal integer greater than or equal to 1. The default shall be equivalent to *n*=1. If *n* is greater than the number of enclosing loops, the outermost enclosing loop shall be exited. Execution shall continue with the command immediately following the loop.

OPTIONS

None.

OPERANDS

None.

STDIN

None.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

None.

STDOUT

None.

STDERR

None.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

0 Successful completion.

>0 The *n* value was not an unsigned decimal integer greater than or equal to 1.

CONSEQUENCES OF ERRORS

None.

APPLICATION USAGE

None.

EXAMPLES

```
for i in * do
    if test –d "$i" then break fi done
```

RATIONALE

In early proposals, consideration was given to expanding the syntax of *break* and *continue* to refer to a label associated with the appropriate loop as a preferable alternative to the *n* method. However, this volume of IEEE Std 1003.1-200x does reserve the namespace of command names ending with a colon. It is anticipated that a future implementation could take advantage of this and provide something like:

```
outofloop: for i in a b c d e
do
    for j in 0 1 2 3 4 5 6 7 8 9
    do
        if test –r "${i}${j}"
        then break outofloop
        fi
    done
done
```

and that this might be standardized after implementation experience is achieved.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14 (on page 2266)

CHANGE HISTORY

None.

2717 **NAME**

2718 colon — null utility

2719 **SYNOPSIS**

2720 `: [`*`argument ...`*`]`

2721 **DESCRIPTION**

2722 This utility shall only expand command *argument*s. It is used when a command is needed, as in
2723 the *then* condition of an **if** command, but nothing is to be done by the command.

2724 **OPTIONS**

2725 None.

2726 **OPERANDS**

2727 None.

2728 **STDIN**

2729 None.

2730 **INPUT FILES**

2731 None.

2732 **ENVIRONMENT VARIABLES**

2733 None.

2734 **ASYNCHRONOUS EVENTS**

2735 None.

2736 **STDOUT**

2737 None.

2738 **STDERR**

2739 None.

2740 **OUTPUT FILES**

2741 None.

2742 **EXTENDED DESCRIPTION**

2743 None.

2744 **EXIT STATUS**

2745 Zero.

2746 **CONSEQUENCES OF ERRORS**

2747 None.

2748 **APPLICATION USAGE**

2749 None.

2750 **EXAMPLES**

2751
```
: ${X=abc}
if     false
then   :
else   echo $X
fi
abc
```

2757 As with any of the special built-ins, the null utility can also have variable assignments and
2758 redirections associated with it, such as:

```
x=y : > z
```

which sets variable *x* to the value *y* (so that it persists after the null utility completes) and creates or truncates file **z**.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14 (on page 2266)

CHANGE HISTORY

None.

NAME

continue — continue for, while, or until loop

SYNOPSIS

```
continue [n]
```

DESCRIPTION

The *continue* utility shall return to the top of the smallest enclosing **for**, **while**, or **until** loop, or to the top of the *n*th enclosing loop, if *n* is specified. This involves repeating the condition list of a **while** or **until** loop or performing the next assignment of a **for** loop, and reexecuting the loop if appropriate.

The value of *n* is a decimal integer greater than or equal to 1. The default shall be equivalent to *n*=1. If *n* is greater than the number of enclosing loops, the outermost enclosing loop shall be used.

OPTIONS

None.

OPERANDS

None.

STDIN

None.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

None.

STDOUT

None.

STDERR

None.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

0 Successful completion.

>0 The *n* value was not an unsigned decimal integer greater than or equal to 1.

CONSEQUENCES OF ERRORS

None.

2807 **APPLICATION USAGE**

2808 None.

2809 **EXAMPLES**

```
2810        for i in *
2811        do
2812            if test –d "$i"
2813            then continue
2814            fi
2815            echo "\"$i\"" is not a directory.
2816        done
```

2817 **RATIONALE**

2818 None.

2819 **FUTURE DIRECTIONS**

2820 None.

2821 **SEE ALSO**

2822 Section 2.14 (on page 2266)

2823 **CHANGE HISTORY**

2824 None.

NAME

dot — execute commands in current environment

SYNOPSIS

```
. file
```

DESCRIPTION

The shell shall execute commands from the *file* in the current environment.

If *file* does not contain a slash, the shell shall use the search path specified by *PATH* to find the directory containing *file*. Unlike normal command search, however, the file searched for by the *dot* utility need not be executable. If no readable file is found, a non-interactive shell shall abort; an interactive shell shall write a diagnostic message to standard error, but this condition shall not be considered a syntax error.

OPTIONS

None.

OPERANDS

None.

STDIN

None.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

None.

STDOUT

None.

STDERR

None.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

Returns the value of the last command executed, or a zero exit status if no command is executed. |

CONSEQUENCES OF ERRORS

None. |

APPLICATION USAGE

None.

EXAMPLES

```
cat foobar
foo=hello bar=world
. foobar
echo $foo $bar
hello world
```

RATIONALE

Some older implementations searched the current directory for the *file*, even if the value of *PATH* disallowed it. This behavior was omitted from this volume of IEEE Std 1003.1-200x due to concerns about introducing the susceptibility to trojan horses that the user might be trying to avoid by leaving *dot* out of *PATH*.

The KornShell version of *dot* takes optional arguments that are set to the positional parameters. This is a valid extension that allows a *dot* script to behave identically to a function.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14 (on page 2266)

CHANGE HISTORY

None.

NAME

eval — construct command by concatenating arguments

SYNOPSIS

```
eval [argument ...]
```

DESCRIPTION

The *eval* utility shall construct a command by concatenating *argument*s together, separating each with a <space>. The constructed command shall be read and executed by the shell.

OPTIONS

None.

OPERANDS

None.

STDIN

None.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

None.

STDOUT

None.

STDERR

None.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

If there are no *argument*s, or only null arguments, *eval* shall return a zero exit status; otherwise, it shall return the exit status of the command defined by the string of concatenated *argument*s separated by spaces.

CONSEQUENCES OF ERRORS

None.

APPLICATION USAGE

None.

EXAMPLES

```
foo=10 x=foo
y='$'$x
echo $y
$foo
eval y='$'$x
echo $y
10
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14 (on page 2266)

CHANGE HISTORY

None.

NAME

exec — execute commands and open, close, or copy file descriptors

SYNOPSIS

```
exec [command [argument ...]]
```

DESCRIPTION

The *exec* utility shall open, close, and/or copy file descriptors as specified by any redirections as part of the command.

If *exec* is specified without *command* or *argument*s, and any file descriptors with numbers greater than 2 are opened with associated redirection statements, it is unspecified whether those file descriptors remain open when the shell invokes another utility. Scripts concerned that child shells could misuse open file descriptors can always close them explicitly, as shown in one of the following examples.

If *exec* is specified with *command*, it shall replace the shell with *command* without creating a new process. If *argument*s are specified, they shall be arguments to *command*. Redirection affects the current shell execution environment.

OPTIONS

None.

OPERANDS

None.

STDIN

None.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

None.

STDOUT

None.

STDERR

None.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

If *command* is specified, *exec* shall not return to the shell; rather, the exit status of the process shall be the exit status of the program implementing *command*, which overlaid the shell. If *command* is not found, the exit status shall be 127. If *command* is found, but it is not an executable utility, the exit status shall be 126. If a redirection error occurs (see Section 2.8.1 (on page 2247)), the shell shall exit with a value in the range 1–125. Otherwise, *exec* shall return a zero exit status.

2973 **CONSEQUENCES OF ERRORS**
2974 None.

2975 **APPLICATION USAGE**
2976 None.

2977 **EXAMPLES**
2978 Open *readfile* as file descriptor 3 for reading:

2979 ```
exec 3< readfile
```

2980 Open *writefile* as file descriptor 4 for writing:

2981 ```
exec 4> writefile
```

2982 Make file descriptor 5 a copy of file descriptor 0:

2983 ```
exec 5<&0
```

2984 Close file descriptor 3:

2985 ```
exec 3<&–
```

2986 Cat the file **maggie** by replacing the current shell with the *cat* utility:

2987 ```
exec cat maggie
```

2988 **RATIONALE**
2989 Most historical implementations were not conformant in that:

2990 ```
foo=bar exec cmd
```

2991 did not pass **foo** to **cmd**.

2992 **FUTURE DIRECTIONS**
2993 None.

2994 **SEE ALSO**
2995 Section 2.14 (on page 2266)

2996 **CHANGE HISTORY**
2997 None.

2998 **NAME**

2999 exit — cause the shell to exit

3000 **SYNOPSIS**

3001 `exit [`*n*`]`

3002 **DESCRIPTION**

3003 The *exit* utility shall cause the shell to exit with the exit status specified by the unsigned decimal
3004 integer *n*. If *n* is specified, but its value is not between 0 and 255 inclusively, the exit status is
3005 undefined.

3006 A *trap* on **EXIT** shall be executed before the shell terminates, except when the *exit* utility is
3007 invoked in that *trap* itself, in which case the shell shall exit immediately.

3008 **OPTIONS**

3009 None.

3010 **OPERANDS**

3011 None.

3012 **STDIN**

3013 None.

3014 **INPUT FILES**

3015 None.

3016 **ENVIRONMENT VARIABLES**

3017 None.

3018 **ASYNCHRONOUS EVENTS**

3019 None.

3020 **STDOUT**

3021 None.

3022 **STDERR**

3023 None.

3024 **OUTPUT FILES**

3025 None.

3026 **EXTENDED DESCRIPTION**

3027 None.

3028 **EXIT STATUS**

3029 The exit status shall be *n*, if specified. Otherwise, the value shall be the exit value of the last
3030 command executed, or zero if no command was executed. When *exit* is executed in a *trap* action,
3031 the last command is considered to be the command that executed immediately preceding the
3032 *trap* action. |

3033 **CONSEQUENCES OF ERRORS**

3034 None. |

APPLICATION USAGE

None.

EXAMPLES

Exit with a *true* value:

```
exit 0
```

Exit with a *false* value:

```
exit 1
```

RATIONALE

As explained in other sections, certain exit status values have been reserved for special uses and should be used by applications only for those purposes:

126 A file to be executed was found, but it was not an executable utility.

127 A utility to be executed was not found.

>128 A command was interrupted by a signal.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14 (on page 2266)

CHANGE HISTORY

None.

NAME

export — set export attribute for variables

SYNOPSIS

```
export name[=word]...
export –p
```

DESCRIPTION

The shell shall give the export attribute to the variables corresponding to the specified *name*s, which shall cause them to be in the environment of subsequently executed commands.

The *export* special built-in shall support the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

When **–p** is specified, *export* shall write to the standard output the names and values of all exported variables, in the following format:

```
"export %s=%s\n", <name>, <value>
```

if *name* is set, and:

```
"export %s\n", <name>
```

if *name* is unset.

The shell shall format the output, including the proper use of quoting, so that it is suitable for reinput to the shell as commands that achieve the same exporting results, except:

1. Read-only variables with values cannot be reset.
2. Variables that were unset at the time they were output need not be reset to the unset state if a value is assigned to the variable between the time the state was saved and the time at which the saved output is reinput to the shell.

When no arguments are given, the results are unspecified.

OPTIONS

None.

OPERANDS

None.

STDIN

None.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

None.

STDOUT

None.

STDERR

None.

3093 **OUTPUT FILES**

3094 None.

3095 **EXTENDED DESCRIPTION**

3096 None.

3097 **EXIT STATUS**

3098 Zero.

3099 **CONSEQUENCES OF ERRORS**

3100 None.

3101 **APPLICATION USAGE**

3102 None.

3103 **EXAMPLES**

3104 Export *PWD* and *HOME* variables:

3105
```
export PWD HOME
```

3106 Set and export the *PATH* variable:

3107
```
export PATH=/local/bin:$PATH
```

3108 Save and restore all exported variables:

3109
```
export -p > temp-file
3110 unset a lot of variables
3111 ... processing
3112 . temp-file
```

3113 **RATIONALE**

3114 Some historical shells use the no-argument case as the functional equivalent of what is required
3115 here with **–p**. This feature was left unspecified because it is not historical practice in all shells,
3116 and some scripts may rely on the now-unspecified results on their implementations. Attempts to
3117 specify the **–p** output as the default case were unsuccessful in achieving consensus. The **–p**
3118 option was added to allow portable access to the values that can be saved and then later restored
3119 using; for example, a *dot* script.

3120 **FUTURE DIRECTIONS**

3121 None.

3122 **SEE ALSO**

3123 Section 2.14 (on page 2266)

3124 **CHANGE HISTORY**

3125 **Issue 6**

3126 IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.

3127 **NAME**

3128 readonly — set read-only attribute for variables

3129 **SYNOPSIS**

3130 `readonly name[=word]...`

3131 `readonly –p`

3132 **DESCRIPTION**

3133 The variables whose *name*s are specified shall be given the *readonly* attribute. The values of
3134 variables with the *readonly* attribute cannot be changed by subsequent assignment, nor can those
3135 variables be unset by the *unset* utility.

3136 The *readonly* special built-in shall support the Base Definitions volume of IEEE Std 1003.1-200x,
3137 Section 12.2, Utility Syntax Guidelines.

3138 When **–p** is specified, *readonly* writes to the standard output the names and values of all read-
3139 only variables, in the following format:

3140 `"readonly %s=%s\n", <name>, <value>`

3141 if *name* is set, and |

3142 `"readonly %s\n", <name>` |

3143 if *name* is unset. |

3144 The shell shall format the output, including the proper use of quoting, so that it is suitable for |
3145 reinput to the shell as commands that achieve the same value and read-only attribute-setting |
3146 results in a shell execution environment in which: |

3147 1. Variables with values at the time they were output do not have the read-only attribute set. |

3148 2. Variables that were unset at the time they were output do not have a value at the time at |
3149 which the saved output is reinput to the shell. |

3150 When no arguments are given, the results are unspecified. |

3151 **OPTIONS**

3152 None.

3153 **OPERANDS**

3154 None.

3155 **STDIN**

3156 None.

3157 **INPUT FILES**

3158 None.

3159 **ENVIRONMENT VARIABLES**

3160 None.

3161 **ASYNCHRONOUS EVENTS**

3162 None.

3163 **STDOUT**

3164 None.

STDERR

None.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

Zero.

CONSEQUENCES OF ERRORS

None.

APPLICATION USAGE

None.

EXAMPLES

```
readonly HOME PWD
```

RATIONALE

Some historical shells preserve the read-only attribute across separate invocations. This volume of IEEE Std 1003.1-200x allows this behavior, but does not require it.

The **–p** option allows portable access to the values that can be saved and then later restored using; for example, a *dot* script. Also see the RATIONALE for *export* (on page 2281) for a description of the no-argument and **–p** output cases and a related example.

Read-only functions were considered, but they were omitted as not being historical practice or particularly useful. Furthermore, functions must not be *readonly* across invocations to preclude *spoofing* (spoofing is the term for the practice of creating a program that acts like a well-known utility with the intent of subverting the real intent of the user) of administrative or security-relevant (or security-conscious) shell scripts.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14 (on page 2266)

CHANGE HISTORY

Issue 6

IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the format when a variable is unset.

3197 **NAME**

3198 return — return from a function

3199 **SYNOPSIS**

3200 `return [`*n*`]`

3201 **DESCRIPTION**

3202 The *return* utility shall cause the shell to stop executing the current function or *dot* script. If the
3203 shell is not currently executing a function or *dot* script, the results are unspecified.

3204 **OPTIONS**

3205 None.

3206 **OPERANDS**

3207 None.

3208 **STDIN**

3209 None.

3210 **INPUT FILES**

3211 None.

3212 **ENVIRONMENT VARIABLES**

3213 None.

3214 **ASYNCHRONOUS EVENTS**

3215 None.

3216 **STDOUT**

3217 None.

3218 **STDERR**

3219 None.

3220 **OUTPUT FILES**

3221 None.

3222 **EXTENDED DESCRIPTION**

3223 None.

3224 **EXIT STATUS**

3225 The value of the special parameter `'?'` shall be set to *n*, an unsigned decimal integer, or to the
3226 exit status of the last command executed if *n* is not specified. If the value of *n* is greater than 255,
3227 the results are undefined. When *return* is executed in a *trap* action, the last command is
3228 considered to be the command that executed immediately preceding the *trap* action.

3229 **CONSEQUENCES OF ERRORS**

3230 None.

3231 **APPLICATION USAGE**

3232 None.

3233 **EXAMPLES**

3234 None.

3235 **RATIONALE**

3236 The behavior of *return* when not in a function or *dot* script differs between the System V shell
3237 and the KornShell. In the System V shell this is an error, whereas in the KornShell, the effect is
3238 the same as *exit*.

The results of returning a number greater than 255 are undefined because of differing practices in the various historical implementations. Some shells AND out all but the low-order 8 bits; others allow larger values, but not of unlimited size.

See the discussion of appropriate exit status values under *exit* (on page 2279).

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14 (on page 2266)

CHANGE HISTORY

None.

NAME

set — set or unset options and positional parameters

SYNOPSIS

XSI `set [–abCefmnuvx][–h][–o option][argument...]`

XSI `set [+abCefmnuvx][+h][+o option][argument...]`

`set ––[argument...]`

`set –o`

`set +o`

DESCRIPTION

If no options or *argument*s are specified, *set* shall write the names and values of all shell variables in the collation sequence of the current locale. Each *name* shall start on a separate line, using the format:

`"%s=%s\n", <name>, <value>`

The *value* string shall be written with appropriate quoting so that it is suitable for reinput to the shell, setting or resetting, as far as possible, the variables that are currently set. Read-only variables cannot be reset; see the description of shell quoting in Section 2.2 (on page 2232).

When options are specified, they shall set or unset attributes of the shell, as described below. When *argument*s are specified, they cause positional parameters to be set or unset, as described below. Setting or unsetting attributes and positional parameters are not necessarily related actions, but they can be combined in a single invocation of *set*.

The *set* special built-in shall support the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines except that options can be specified with either a leading hyphen (meaning enable the option) or plus sign (meaning disable it).

Implementations shall support the options in the following list in both their hyphen and plus-sign forms. These options can also be specified as options to *sh*.

–a When this option is on, the export attribute shall be set for each variable to which an assignment is performed; see the Base Definitions volume of IEEE Std 1003.1-200x, Section 4.21, Variable Assignment. If the assignment precedes a utility name in a command, the export attribute shall not persist in the current execution environment after the utility completes, with the exception that preceding one of the special built-in utilities causes the export attribute to persist after the built-in has completed. If the assignment does not precede a utility name in the command, or if the assignment is a result of the operation of the *getopts* or *read* utilities, the export attribute shall persist until the variable is unset.

–b This option shall be supported if the implementation supports the User Portability Utilities option. It shall cause the shell to notify the user asynchronously of background job completions. The following message is 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 also shall 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 Section 2.9.3.1 (on page 2252). Asynchronous notification shall not be enabled by default.

–C (Uppercase C.) Prevent existing files from being overwritten by the shell's '>' redirection operator (see Section 2.7.2 (on page 2245)); the ">|" redirection operator shall override this *noclobber* option for an individual file.

–e When this option is on, if a simple command fails for any of the reasons listed in Section 2.8.1 (on page 2247) or returns an exit status value >0, and is not part of the compound list following a **while**, **until**, or **if** keyword, and is not a part of an AND or OR list, and is not a pipeline preceded by the **!** reserved word, then the shell shall immediately exit.

–f The shell shall disable pathname expansion.

XSI **–h** Locate and remember utilities invoked by functions as those functions are defined (the utilities are normally located when the function is executed).

–m This option shall be supported if the implementation supports the User Portability Utilities 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 error to that effect, formatted as described by the *jobs* utility. 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. This option is enabled by default for interactive shells.

–n The shell shall read commands but does not execute them; this can be used to check for shell script syntax errors. An interactive shell may ignore this option.

–o Write the current settings of the options to standard output in an unspecified format.

+o Write the current option settings to standard output in a format that is suitable for reinput to the shell as commands that achieve the same options settings.

–o *option*
This option is supported if the system supports the User Portability Utilities option. It shall 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**.

errexit Equivalent to **–e**.

ignoreeof Prevent an interactive shell from exiting on end-of-file. This setting prevents accidental logouts when <control>-D is entered. A user shall explicitly *exit* to leave the interactive shell.

monitor Equivalent to **–m**. This option is supported if the system supports the User Portability Utilities option.

noclobber Equivalent to **–C** (uppercase C).

noglob Equivalent to **–f**.

noexec Equivalent to **–n**.

nolog Prevent the entry of function definitions into the command history; see **Command History List** (on page 3052).

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.

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

xtrace Equivalent to **–x**.

–u The shell shall write a message to standard error when it tries to expand a variable that is not set and immediately exit. An interactive shell shall not exit.

–v The shell shall write its input to standard error as it is read.

–x The shell shall write to standard error a trace for each command after it expands the command and before it executes it. It is unspecified whether the command that turns tracing off is traced.

The default for all these options shall be off (unset) unless the shell was invoked with them on; see *sh*.

The remaining arguments shall be assigned in order to the positional parameters. The special parameter '#' shall be set to reflect the number of positional parameters. All positional parameters shall be unset before any new values are assigned.

The special argument "––" immediately following the *set* command name can be used to delimit the arguments if the first argument begins with '+' or '–', or to prevent inadvertent listing of all shell variables when there are no arguments. The command *set*–– without *argument* shall unset all positional parameters and set the special parameter '#' to zero.

OPTIONS

None.

OPERANDS

None.

STDIN

None.

INPUT FILES

None.

3375 **ENVIRONMENT VARIABLES**

3376 None.

3377 **ASYNCHRONOUS EVENTS**

3378 None.

3379 **STDOUT**

3380 None.

3381 **STDERR**

3382 None.

3383 **OUTPUT FILES**

3384 None.

3385 **EXTENDED DESCRIPTION**

3386 None.

3387 **EXIT STATUS**

3388 Zero.

3389 **CONSEQUENCES OF ERRORS**

3390 None.

3391 **APPLICATION USAGE**

3392 None.

3393 **EXAMPLES**

3394 Write out all variables and their values:

3395 `set`

3396 Set $1, $2, and $3 and set "$#" to 3:

3397 `set c a b`

3398 Turn on the **–x** and **–v** options:

3399 `set –xv`

3400 Unset all positional parameters:

3401 `set ––`

3402 Set $1 to the value of **–x**, even if **x** begins with `'–'` or `'+'`:

3403 `set –– "$x"`

3404 Set the positional parameters to the expansion of **x**, even if **x** expands with a leading `'–'` or `'+'`:

3405 `set –– $x`

3406 **RATIONALE**

3407 The *set* –– form is listed specifically in the SYNOPSIS even though this usage is implied by the
3408 Utility Syntax Guidelines. The explanation of this feature removes any ambiguity about whether
3409 the *set* –– form might be misinterpreted as being equivalent to *set* without any options or
3410 arguments. The functionality of this form has been adopted from the KornShell. In System V, *set*
3411 –– only unsets parameters if there is at least one argument; the only way to unset all parameters
3412 is to use *shift*. Using the KornShell version should not affect System V scripts because there
3413 should be no reason to issue it without arguments deliberately; if it were issued as, for example:

3414 `set –– "$@"`

and there were in fact no arguments resulting from "$@", unsetting the parameters would have no result.

The *set* + form in early proposals was omitted as being an unnecessary duplication of *set* alone and not widespread historical practice.

The *noclobber* option was changed to allow *set* –**C** as well as the *set* –**o** *noclobber* option. The single-letter version was added so that the historical "$–" paradigm would not be broken; see Section 2.5.2 (on page 2235).

The –**h** flag is related to command name hashing and is only required on XSI-conformant systems.

The following *set* flags were omitted intentionally with the following rationale:

–**k** The –**k** flag was originally added by the author of the Bourne shell to make it easier for users of pre-release versions of the shell. In early versions of the Bourne shell the construct *set name=value*, had to be used to assign values to shell variables. The problem with –**k** is that the behavior affects parsing, virtually precluding writing any compilers. To explain the behavior of –**k**, it is necessary to describe the parsing algorithm, which is implementation-defined. For example:

```
set –k; echo name=value
```

and:

```
set x––k
echo name=value
```

behave differently. The interaction with functions is even more complex. What is more, the –**k** flag is never needed, since the command line could have been reordered.

–**t** The –**t** flag is hard to specify and almost never used. The only known use could be done with here-documents. Moreover, the behavior with *ksh* and *sh* differs. The reference page says that it exits after reading and executing one command. What is one command? If the input is *date*;*date*, *sh* executes both *date* commands while *ksh* does only the first.

Consideration was given to rewriting *set* to simplify its confusing syntax. A specific suggestion was that the *unset* utility should be used to unset options instead of using the non-*getopt*( )-able +*option* syntax. However, the conclusion was reached that the historical practice of using +*option* was satisfactory and that there was no compelling reason to modify such widespread historical practice.

The –**o** option was adopted from the KornShell to address user needs. In addition to its generally friendly interface, –**o** is needed to provide the *vi* command line editing mode, for which historical practice yields no single-letter option name. (Although it might have been possible to invent such a letter, it was recognized that other editing modes would be developed and –**o** provides ample name space for describing such extensions.)

Historical implementations are inconsistent in the format used for –**o** option status reporting. The **+o** format without an option-argument was added to allow portable access to the options that can be saved and then later restored using, for instance, a dot script.

Historically, *sh* did trace the command *set* **+x**, but *ksh* did not.

The *ignoreeof* setting prevents accidental logouts when the end-of-file character (typically <control>-D) is entered. A user shall explicitly *exit* to leave the interactive shell.

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

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* asynchronously informs the user of changes in the states of the specified jobs.

To add asynchronous notification to the POSIX shell, neither the KornShell extensions to *trap*, nor the C shell *notify* environment variable seemed appropriate (*notify* 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.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14 (on page 2266)

CHANGE HISTORY

Issue 6

The obsolescent *set* command name followed by `'–'` has been removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The *nolog* option is added to *set* –**o**.

NAME

shift — shift positional parameters

SYNOPSIS

```
shift [n]
```

DESCRIPTION

The positional parameters shall be shifted. Positional parameter 1 shall be assigned the value of parameter (1+*n*), parameter 2 shall be assigned the value of parameter (2+*n*), and so on. The parameters represented by the numbers `"$#"` down to `"$#–n+1"` shall be unset, and the parameter `'#'` is updated to reflect the new number of positional parameters.

The value *n* shall be an unsigned decimal integer less than or equal to the value of the special parameter `'#'`. If *n* is not given, it shall be assumed to be 1. If *n* is 0, the positional and special parameters are not changed.

OPTIONS

None.

OPERANDS

None.

STDIN

None.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

None.

STDOUT

None.

STDERR

None.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The exit status is >0 if *n*>$#; otherwise, it is zero.

CONSEQUENCES OF ERRORS

None.

3523 **APPLICATION USAGE**

3524 None.

3525 **EXAMPLES**

3526
```
$ set a b c d e
```
3527
```
$ shift 2
```
3528
```
$ echo $*
```
3529
```
c d e
```

3530 **RATIONALE**

3531 None.

3532 **FUTURE DIRECTIONS**

3533 None.

3534 **SEE ALSO**

3535 Section 2.14 (on page 2266)

3536 **CHANGE HISTORY**

3537 None.

NAME

times — write process times

SYNOPSIS

```
times
```

DESCRIPTION

Write the accumulated user and system times for the shell and for all of its child processes, in the following POSIX locale format:

```
"%dm%fs %dm%fs\n%dm%fs %dm%fs\n", <shell user minutes>,
    <shell user seconds>, <shell system minutes>,
    <shell system seconds>, <children user minutes>,
    <children user seconds>, <children system minutes>,
    <children system seconds>
```

The four pairs of times shall correspond to the members of the **<sys/times.h>** **tms** structure (defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 13, Headers) as returned by *times*( ): *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime*, respectively.

OPTIONS

None.

OPERANDS

None.

STDIN

None.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

None.

STDOUT

None.

STDERR

None.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

Zero.

CONSEQUENCES OF ERRORS

None.

APPLICATION USAGE

None.

EXAMPLES

```
$ times
0m0.43s 0m1.11s
8m44.18s 1m43.23s
```

RATIONALE

The *times* special built-in from the Single UNIX Specification is now required for all conforming shells.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14 (on page 2266)

CHANGE HISTORY

None.

3592 **NAME**

3593 trap — trap signals

3594 **SYNOPSIS**

3595 `trap [`*`action condition ...`*`]`

3596 **DESCRIPTION**

3597 If *action* is `'-'`, the shell shall reset each *condition* to the default value. If *action* is null (`""`), the
3598 shell shall ignore each specified *condition* if it arises. Otherwise, the argument *action* shall be read
3599 and executed by the shell when one of the corresponding conditions arises. The action of *trap*
3600 shall override a previous action (either default action or one explicitly set). The value of `"$?"`
3601 after the *trap* action completes shall be the value it had before *trap* was invoked.

3602 The condition can be EXIT, 0 (equivalent to EXIT), or a signal specified using a symbolic name,
3603 without the SIG prefix, as listed in the tables of signal names in the **<signal.h>** header defined in
3604 the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 13, Headers; for example, HUP,
3605 INT, QUIT, TERM. Implementations may permit lowercase signal names or names with the SIG
3606 prefix as an extension. Setting a trap for SIGKILL or SIGSTOP produces undefined results.

3607 The environment in which the shell executes a *trap* on EXIT shall be identical to the environment
3608 immediately after the last command executed before the *trap* on EXIT was taken.

3609 Each time *trap* is invoked, the *action* argument shall be processed in a manner equivalent to:

3610
```
eval "$action"
```

3611 Signals that were ignored on entry to a non-interactive shell cannot be trapped or reset, although
3612 no error need be reported when attempting to do so. An interactive shell may reset or catch
3613 signals ignored on entry. Traps shall remain in place for a given shell until explicitly changed
3614 with another *trap* command.

3615 When a subshell is entered, traps that are not being ignored are set to the default actions. This
3616 does not imply that the *trap* command cannot be used within the subshell to set new traps.

3617 The *trap* command with no arguments shall write to standard output a list of commands
3618 associated with each condition. The format shall be:

3619 `"trap -- %s %s ...\n", <`*`action`*`>, <`*`condition`*`> ...`

3620 The shell shall format the output, including the proper use of quoting, so that it is suitable for
3621 reinput to the shell as commands that achieve the same trapping results. For example:

3622
```
save_traps=$(trap)
...
eval "$save_traps"
```

3625 XSI XSI-conformant systems also allow numeric signal numbers for the conditions corresponding to
3626 the following signal names:

Signal Number	Signal Name
1	SIGHUP
2	SIGINT
3	SIGQUIT
6	SIGABRT
9	SIGKILL
14	SIGALRM
15	SIGTERM

The *trap* special built-in shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

OPTIONS

None.

OPERANDS

None.

STDIN

None.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

None.

STDOUT

None.

STDERR

None.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

If the trap name or number is invalid, a non-zero exit status shall be returned; otherwise, zero shall be returned. For both interactive and non-interactive shells, invalid signal names or numbers shall not be considered a syntax error and do not cause the shell to abort.

CONSEQUENCES OF ERRORS

None.

APPLICATION USAGE

None.

EXAMPLES

Write out a list of all traps and actions:

```
trap
```

Set a trap so the *logout* utility in the directory referred to by the *HOME* environment variable executes when the shell terminates:

```
trap '$HOME/logout' EXIT
```

or:

```
trap '$HOME/logout' 0
```

Unset traps on INT, QUIT, TERM, and EXIT:

```
trap - INT QUIT TERM EXIT
```

RATIONALE

Implementations may permit lowercase signal names as an extension. Implementations may also accept the names with the SIG prefix; no known historical shell does so. The *trap* and *kill* utilities in this volume of IEEE Std 1003.1-200x are now consistent in their omission of the SIG prefix for signal names. Some *kill* implementations do not allow the prefix, and *kill* **-l** lists the signals without prefixes.

Trapping SIGKILL or SIGSTOP is syntactically accepted by some historical implementations, but it has no effect. Portable POSIX applications cannot attempt to trap these signals.

The output format is not historical practice. Since the output of historical *trap* commands is not portable (because numeric signal values are not portable) and had to change to become so, an opportunity was taken to format the output in a way that a shell script could use to save and then later reuse a trap if it wanted.

The KornShell uses an **ERR** trap that is triggered whenever *set* **-e** would cause an exit. This is allowable as an extension, but was not mandated, as other shells have not used it.

The text about the environment for the EXIT trap invalidates the behavior of some historical versions of interactive shells which, for example, close the standard input before executing a trap on 0. For example, in some historical interactive shell sessions the following trap on 0 would always print "--":

```
trap 'read foo; echo "-$foo-"' 0
```

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14 (on page 2266)

CHANGE HISTORY

Issue 6

XSI-conforming implementations provide the mapping of signal names to numbers given above (previously this had been marked obsolescent). Other implementations need not provide this optional mapping.

NAME

unset — unset values and attributes of variables and functions

SYNOPSIS

```
unset [–fv] name ...
```

DESCRIPTION

Each variable or function specified by *name* shall be unset.

If **–v** is specified, *name* refers to a variable name and the shell shall unset it and remove it from the environment. Read-only variables cannot be unset.

If **–f** is specified, *name* refers to a function and the shell shall unset the function definition.

If neither **–f** nor **–v** is specified, *name* refers to a variable; if a variable by that name does not exist, it is unspecified whether a function by that name, if any, shall be unset.

Unsetting a variable or function that was not previously set shall not be considered an error and does not cause the shell to abort.

The *unset* special built-in shall support the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

Note that:

```
VARIABLE=
```

is not equivalent to an *unset* of **VARIABLE**; in the example, **VARIABLE** is set to `""`. Also, the variables that can be *unset* should not be misinterpreted to include the special parameters (see Section 2.5.2 (on page 2235)).

OPTIONS

None.

OPERANDS

None.

STDIN

None.

INPUT FILES

None.

ENVIRONMENT VARIABLES

None.

ASYNCHRONOUS EVENTS

None.

STDOUT

None.

STDERR

None.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

0 All *name* operands were successfully unset.

>0 At least one *name* could not be unset.

CONSEQUENCES OF ERRORS

None.

APPLICATION USAGE

None.

EXAMPLES

Unset *VISUAL* variable:

```
unset –v VISUAL
```

Unset the functions **foo** and **bar**:

```
unset –f foo bar
```

RATIONALE

Consideration was given to omitting the **–f** option in favor of an *unfunction* utility, but the standard developers decided to retain historical practice.

The **–v** option was introduced because System V historically used one name space for both variables and functions. When *unset* is used without options, System V historically unset either a function or a variable, and there was no confusion about which one was intended. A portable POSIX application can use *unset* without an option to unset a variable, but not a function; the **–f** option must be used.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14 (on page 2266)

CHANGE HISTORY

None.

3770

Chapter 3

Batch Environment Services

BE This chapter describes the services and utilities that shall be implemented on all systems that claim conformance to the Batch Environment option. This functionality is dependent on support of this option (and the rest of this section is not further shaded for this option).

3.1 General Concepts

3.1.1 Batch Client-Server Interaction

Batch jobs are created and managed by batch servers. A batch client interacts with a batch server to access batch services on behalf of the user. In order to use batch services, a user must have access to a batch client.

A batch server is a computational entity, such as a daemon process, that provides batch services. Batch servers route, queue, modify, and execute batch jobs on behalf of batch clients.

The batch utilities described in this volume of IEEE Std 1003.1-200x (and listed in Table 3-1) are clients of batch services; they allow users to perform actions on the job such as creating, modifying, and deleting batch jobs from a shell command line. Although these batch utilities may be said to accomplish certain services, they actually obtain services on behalf of a user by means of requests to batch servers.

Table 3-1 Batch Utilities

qalter	*qmove*	*qrls*	*qstat*
qdel	*qmsg*	*qselect*	*qsub*
qhold	*qrerun*	*qsig*	

Client-server interaction takes place by means of the batch requests defined in this chapter. Because direct access to batch jobs and queues is limited to batch servers, clients and servers of different implementations can interoperate, since dependencies on private structures for batch jobs and queues are limited to batch servers. Also, batch servers may be clients of other batch servers.

3.1.2 Batch Queues

Two types of batch queue are described: *routing queues* and *execution queues*. When a batch job is placed in a routing queue, it is a candidate for routing. A batch job is removed from routing queues under the following conditions:

- The batch job has been routed to another queue.
- The batch job has been deleted from the batch queue.
- The batch job has been aborted.

When a batch job is placed in an execution queue, it is a candidate for execution.

A batch job is removed from an execution queue under the following conditions:

- The batch job has been executed and exited.

- The batch job has been aborted.
- The batch job has been deleted from the batch queue.
- The batch job has been moved to another queue.

Access to a batch queue is limited to the batch server that manages the batch queue. Clients never access a batch queue or a batch job directly, either to read or write information; all client access to batch queues or jobs takes place through batch servers.

3.1.3 Batch Job Creation

When a batch server creates a batch job on behalf of a client, it shall assign a batch job identifier to the job. A batch job identifier consists of both a sequence number that is unique among the sequence numbers issued by that server and the name of the server. Since the batch server name is unique within a name space, the job identifier is likewise unique within the name space.

The batch server that creates a batch job shall return the batch server-assigned job identifier to the client that requested the job creation. If the batch server routes or moves the job to another server, it sends the job identifier with the job. Once assigned, the job identifier of a batch job shall never change.

3.1.4 Batch Job Tracking

Since a batch job may be moved after creation, the batch server name component of the job identifier need not indicate the location of the job. An implementation may provide a batch job tracking mechanism, in which case the user generally does not need to know the location of the job. However, an implementation need not provide a batch job tracking mechanism, in which case the user must find routed jobs by probing the possible destinations.

3.1.5 Batch Job Routing

To route a batch job, a batch server either moves the job to some other queue that is managed by the batch server, or requests that some other batch server accept the job.

Each routing queue has one or more queues to which it can route batch jobs. The batch server administrator creates routing queues.

A batch server may route a batch job from a routing queue to another routing queue. Batch servers shall prevent or otherwise handle cases of circular routing paths. As a deferred service, a batch server routes jobs from the routing queues that it manages. The algorithm by which a batch server selects a batch queue to which to route a batch job is implementation-defined.

A batch job need not be eligible for routing to all the batch queues fed by the routing queue from which it is routed. A batch server that has been asked to accept the job may reject the request if the job requires resources that are unavailable to that batch server, or if the client is not authorized to access the batch server.

Batch servers may route high-priority jobs before low-priority jobs, but, on other than overloaded systems, the effect may be imperceptible to the user. If all the batch servers fed by a routing queue reject requests to accept the job for reasons that are permanent, the batch server that manages the job shall abort the job. If all or some rejections are temporary, the batch server should try to route the job again at some later point.

The reasons for rejecting a batch job are implementation-defined. The reasons for which the routing should be retried later and the reasons for which the job should be aborted are also implementation-defined.

3.1.6 Batch Job Execution

To execute a batch job is to create a session leader (a process) that runs the shell program indicated by the *Shell_Path* attribute of the job. The script shall be passed to the program as its standard input. An implementation may pass the script to the program by other implementation-defined means. At the time a batch job begins execution, it is defined to enter the RUNNING state. The primary program that is executed by a batch job is typically, though not necessarily, a shell program.

A batch server shall execute eligible jobs as a deferred service—no client request is necessary once the batch job is created and eligible. However, the attributes of a batch job, such as the job hold type, may render the job ineligible. A batch server shall scan the execution queues that it manages for jobs that are eligible for execution. The algorithm by which the batch server selects eligible jobs for execution is implementation-defined.

As part of creating the process for the batch job, the batch server shall open the standard output and standard error streams of the session.

The attributes of a batch job may indicate that the batch server executing the job shall send mail to a list of users at the time it begins execution of the job.

3.1.7 Batch Job Exit

When the session leader of an executing job terminates, the job exits. As part of exiting a batch job, the batch server that manages the job shall remove the job from the batch queue in which it resides. The server shall transfer output files of the job to a location described by the attributes of the job.

The attributes of a batch job may indicate that the batch server managing the job shall send mail to a list of users at the time the job exits.

3.1.8 Batch Job Abort

A batch server shall abort jobs for which a required deferred service cannot be performed. The attributes of a batch job may indicate that the batch server that aborts the job shall send mail to a list of users at the time it aborts the job.

3.1.9 Batch Authorization

Clients, such as the batch environment utilities (marked BE), access batch services by means of requests to one or more batch servers. To acquire the services of any given batch server, the user identifier under which the client runs must be authorized to use that batch server.

The user with an associated user name that creates a batch job shall own the job and can perform actions such as read, modify, delete, and move.

A user identifier of the same value at a different host need not be the same user. For example, user name *smith* at host **alpha** may or may not represent the same person as user name *smith* at host **beta**. Likewise, the same person may have access to different user names on different hosts.

An implementation may optionally provide an authorization mechanism that permits one user name to access jobs under another user name.

A process on a client host may be authorized to run processes under multiple user names at a batch server host. Where appropriate, the utilities defined in this volume of IEEE Std 1003.1-200x provide a means for a user to choose from among such user names when creating or modifying a batch job.

3.1.10 Batch Administration

The processing of a batch job by a batch server is affected by the attributes of the job. The processing of a batch job may also be affected by the attributes of the batch queue in which the job resides and by the status of the batch server that manages the job. See also the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.43, Batch Administrator and the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.58, Batch Operator.

3.1.11 Batch Notification

Whereas batch servers are persistent entities, clients are often transient. For example, the *qsub* utility creates a batch job and exits. For this reason, batch servers notify users of batch job events by sending mail to the user that owns the job, or to other designated users.

3.2 Batch Services

The presence of Batch Environment option services is indicated by the configuration variable POSIX2_PBS. A conforming batch server provides services as defined in this section.

A batch server shall provide batch services in two ways:

1. The batch server provides a service at the request of a client.
2. The batch server provides a deferred service as a result of a change in conditions monitored by the batch server.

If a batch server cannot complete a request, it shall reject the request. If a batch server cannot complete a deferred service for a batch job, the batch server shall abort the batch job. Table 3-2 (on page 2307) is a summary of environment variables that shall be supported by an implementation of the batch server and utilities.

Table 3-2 Environment Variable Summary

Variable	Description
PBS_DPREFIX	Defines the directive prefix (see *qsub*)
PBS_ENVIRONMENT	Batch Job is batch or interactive (see Section 3.2.2.1 (on page 2308))
PBS_JOBID	The *job_identifier* attribute of job (see Section 3.2.3.8 (on page 2320))
PBS_JOBNAME	The *job_name* attribute of job (see Section 3.2.3.8 (on page 2320))
PBS_O_HOME	Defines the *HOME* of the batch client (see *qsub*)
PBS_O_HOST	Defines the host name of the batch client (see *qsub*)
PBS_O_LANG	Defines the *LANG* of the batch client (see *qsub*)
PBS_O_LOGNAME	Defines the *LOGNAME* of the batch client (see *qsub*)
PBS_O_MAIL	Defines the *MAIL* of the batch client (see *qsub*)
PBS_O_PATH	Defines the *PATH* of the batch client (see *qsub*)
PBS_O_QUEUE	Defines the submit queue of the batch client (see *qsub*)
PBS_O_SHELL	Defines the *SHELL* of the batch client (see *qsub*)
PBS_O_TZ	Defines the *TZ* of the batch client (see *qsub*)
PBS_O_WORKDIR	Defines the working directory of the batch client (see *qsub*)
PBS_QUEUE	Defines the initial execution queue (see Section 3.2.2.1 (on page 2308))

3.2.1 Batch Job States

A batch job shall always be in one of the following states: QUEUED, RUNNING, HELD, WAITING, EXITING, or TRANSITING. The state of a batch job determines the types of requests that the batch server that manages the batch job can accept for the batch job. A batch server shall change the state of a batch job either in response to service requests from clients or as a result of deferred services, such as job execution or job routing.

A batch job that is in the QUEUED state resides in a queue but is still pending either execution or routing, depending on the queue type.

A batch server that queues a batch job in a routing queue shall put the batch job in the QUEUED state. A batch server that puts a batch job in an execution queue, but has not yet executed the batch job, shall put the batch job in the QUEUED state. A batch job that resides in an execution queue and is executing is defined to be in the RUNNING state. While a batch job is in the RUNNING state, a session leader is associated with the batch job.

A batch job that resides in an execution queue, but is ineligible to run because of a hold attribute, is defined to be in the HELD state.

A batch job that is not held, but must wait until a future date and time before executing, is defined to be in the WAITING state.

When the session leader associated with a running job exits, the batch job shall be placed in the EXITING state.

A batch job for which the session leader has terminated is defined to be in the EXITING state, and the batch server that manages such a batch job cannot accept job modification requests that affect the batch job. While a batch job is in the EXITING state, the batch server that manages the batch job is staging output files and notifying clients of job completion. Once a batch job has exited, it no longer exists as an object managed by a batch server.

A batch job that is being moved from a routing queue to another queue is defined to be in the TRANSITING state.

When a batch job in a routing queue has been selected to be moved to a new destination, then the batch job shal be in either the QUEUED state or the TRANSITING state, depending on the batch server implementation.

Batch jobs with either a *Execution_Time* attribute value set in the future or a *Hold_Types* attribute of value not equal to NO_HOLD, or both, may be routed or held in the routing queue. The treatment of jobs with the *Execution_Time* or *Hold_Types* attributes in a routing queue is implementation-defined.

When a batch job in a routing queue has not been selected to be moved to a new destination and the batch job has a *Hold_Types* attribute value of other than NO_HOLD, then the job should be in the HELD state.

Note: The effect of a hold upon a batch job in a routing queue is implementation-defined. The implementation should use the state that matches whether the batch job can route with a hold or not.

When a batch job in a routing queue has not been selected to be moved to a new destination and the batch job has:

- A *Hold_Types* attribute value of NO_HOLD
- An *Execution_Time* attribute in the past

then the batch job shall be in the QUEUED state.

When a batch job in a routing queue has not been selected to be moved to a new destination and the batch job has:

- A *Hold_Types* attribute value of NO_HOLD
- An *Execution_Time* attribute in the future

then the batch job may be in the WAITING state.

Note: The effect of a future execution time upon a batch job in a routing queue is implementation-defined. The implementation should use the state that matches whether the batch job can route with a hold or not.

Table 3-3 (on page 2309) describes the next state of a batch job, given the current state of the batch job and the type of request. Table 3-4 (on page 2310) describes the response of a batch server to a request, given the current state of the batch job and the type of request.

3.2.2 Deferred Batch Services

This section describes the deferred services performed by batch servers: job execution, job routing, job exit, job abort, and the rerunning of jobs after a restart.

3.2.2.1 Batch Job Execution

To execute a batch job is to create a session leader (a process) that runs the shell program indicated by the *Shell_Path_List* attribute of the batch job. The script is passed to the program as its standard input. An implementation may pass the script to the program by other implementation-defined means. At the time a batch job begins execution, it is defined to enter the RUNNING state.

Table 3-3 Next State Table

Request Type	Current State						
	X	**Q**	**R**	**H**	**W**	**E**	**T**
Queue Batch Job Request	Q	e	e	e	e	e	e
Modify Batch Job Request	e	Q	R	H	W	e	T
Delete Batch Job Request	e	X	E	X	X	E	X
Batch Job Message Request	e	Q	R	H	W	E	T
Rerun Batch Job Request	e	e	Q	e	e	e	e
Signal Batch Job Request	e	e	R	H	W	e	e
Batch Job Status Request	e	Q	R	H	W	E	T
Batch Queue Status Request	X	Q	R	H	W	E	T
Server Status Request	X	Q	R	H	W	E	T
Select Batch Jobs Request	X	Q	R	H	W	E	T
Move Batch Job Request	e	Q	R	H	W	e	T
Hold Batch Job Request	e	H	R / H	H	H	e	T
Release Batch Job Request	Q	R	Q / W / H	W	e	T	
Server Shutdown Request	X	Q	Q	H	W	E	T
Locate Batch Job Request	e	Q	R	H	W	E	T

Legend

X Nonexistent

Q QUEUED

R RUNNING

H HELD

W WAITING

E EXITING

T TRANSITING

e Error

A batch server that has an execution queue containing jobs is said to own the queue and manage the batch jobs in that queue. A batch server that has been started shall execute the batch jobs in the execution queues owned by the batch server. The batch server shall schedule for execution those jobs in the execution queues that are in the QUEUED state. The algorithm for scheduling jobs is implementation-defined.

A batch server that executes a batch job shall create, in the environment of the session leader of the batch job, an environment variable named *PBS_ENVIRONMENT*, the value of which is the string PBS_BATCH encoded in the portable character set.

A batch server that executes a batch job shall create, in the environment of the session leader of the batch job, an environment variable named *PBS_QUEUE*, the value of which is the name of the execution queue of the batch job encoded in the portable character set.

To rerun a batch job is to requeue a batch job that is currently executing and then kill the session leader of the executing job by sending a SIGKILL prior to completion; see Section 3.2.3.11 (on page 2322). A batch server that reruns a batch job shall append the standard output and standard error files of the batch job to the corresponding files of the previous execution, if they exist, with appropriate annotation. If either file does not exist, that file shall be created as in normal execution.

Table 3-4 Results / Output Table

Request Type	Current State						
	X	**Q**	**R**	**H**	**W**	**E**	**T**
Queue Batch Job Request	O	e	e	e	e	e	e
Modify Batch Job Request	e	O	e	O	O	e	e
Delete Batch Job Request	e	O	O	O	O	e	O
Batch Job Message Request	e	e	O	e	e	e	e
Rerun Batch Job Request	e	e	O	e	e	e	e
Signal Batch Job Request	e	e	O	e	e	e	e
Batch Job Status Request	e	O	O	O	O	O	O
Batch Queue Status Request	O	O	O	O	O	O	O
Server Status Request	O	O	O	O	O	O	O
Select Batch Job Request	e	O	O	O	O	O	O
Move Batch Job Request	e	O	O	O	O	e	e
Hold Batch Job Request	e	O	O	O	O	e	e
Release Batch Job Request	e	O	e	O	O	e	e
Server Shutdown Request	O	O	e	O	O	e	e
Locate Batch Job Request	e	O	O	O	O	O	O

Legend

O OK

e Error message

The execution of a batch job by a batch server shall be controlled by job, queue, and server attributes, as defined in this section.

Account_Name Attribute

Batch accounting is an optional feature of batch servers. If a batch server implements accounting, the statements in this section apply and the configuration variable POSIX2_PBS_ACCOUNTING shall be set to 1.

A batch server that executes a batch job shall charge the account named in the *Account_Name* attribute of the batch job for resources consumed by the batch job.

If the *Account_Name* attribute of the batch job is absent from the batch job attribute list or is altered while the batch job is in execution, the batch server action is implementation-defined.

Checkpoint Attribute

Batch checkpointing is an optional feature of batch servers. If a batch server implements checkpointing, the statements in this section apply and the configuration variable POSIX2_PBS_CHECKPOINT shall be set to 1.

There are two attributes associated with the checkpointing feature: *Checkpoint* and *Minimum_Cpu_Interval*. *Checkpoint* is a batch job attribute, while *Minimum_Cpu_Interval* is a queue attribute. An implementation that does not support checkpointing shall support the *Checkpoint* job attribute to the extent that the batch server shall maintain and pass this attribute to other servers.

The behavior of a batch server that executes a batch job for which the value of the *Checkpoint* attribute is CHECKPOINT_UNSPECIFIED is implementation-defined. A batch server that executes a batch job for which the value of the *Checkpoint* attribute is NO_CHECKPOINT shall

not checkpoint the batch job.

A batch server that executes a batch job for which the value of the *Checkpoint* attribute is CHECKPOINT_AT_SHUTDOWN shall checkpoint the batch job only when the batch server accepts a request to shut down during the time when the batch job is in the RUNNING state.

A batch server that executes a batch job for which the value of the *Checkpoint* attribute is CHECKPOINT_AT_MIN_CPU_INTERVAL shall checkpoint the batch job at the interval specified by the *Minimum_Cpu_Interval* attribute of the queue for which the batch job has been selected. The *Minimum_Cpu_Interval* attribute shall be specified in units of CPU minutes.

A batch server that executes a batch job for which the value of the *Checkpoint* attribute is an unsigned integer shall checkpoint the batch job at an interval that is the value of either the *Checkpoint* attribute, or the *Minimum_Cpu_Interval* attribute of the queue for which the batch job has been selected, whichever is greater. Both intervals shall be in units of CPU minutes. When the *Minimum_Cpu_Interval* attribute is greater than the *Checkpoint* attribute, the batch job shall write a warning message to the standard error stream of the batch job.

Error_Path Attribute

The *Error_Path* attribute of a running job cannot be changed by a *Modify Batch Job Request*. When the *Join_Path* attribute of the batch job is set to the value FALSE and the *Keep_Files* attribute of the batch job does not contain the value KEEP_STD_ERROR, a batch server that executes a batch job shall perform one of the following actions:

- Set the standard error stream of the session leader of the batch job to the path described by the value of the *Error_Path* attribute of the batch job.
- Buffer the standard error of the session leader of the batch job until completion of the batch job, and when the batch job exits return the contents to the destination described by the value of the *Error_Path* attribute of the batch job.

Applications shall not rely on having access to the standard error of a batch job prior to the completion of the batch job.

When the *Error_Path* attribute does not specify a host name, then the batch server shall retain the standard error of the batch job on the host of execution.

When the *Error_Path* attribute does specify a host name and the *Keep_Files* attribute does not contain the value KEEP_STD_ERROR, then the final destination of the standard error of the batch job shall be on the host whose host name is specified.

If the path indicated by the value of the *Error_Path* attribute of the batch job is a relative path, the batch server shall expand the path relative to the home directory of the user on the host to which the file is being returned.

When the batch server buffers the standard error of the batch job and the file cannot be opened for write upon completion of the batch job, then the server shall place the standard error in an implementation-defined location and notify the user of the location via mail. It shall be possible for the user to process this mail using the *mailx* utility.

If a batch server that does not buffer the standard error cannot open the standard error path of the batch job for write access, then the batch server shall abort the batch job.

Execution_Time Attribute

A batch server shall not execute a batch job before the time represented by the value of the *Execution_Time* attribute of the batch job. The *Execution_Time* attribute is defined in seconds since the Epoch.

Hold_Types Attribute

A batch server shall support the following hold types:

s Can be set or released by a user with at least a privilege level of batch administrator (SYSTEM).

o Can be set or released by a user with at least a privilege level of batch operator (OPERATOR).

u Can be set or released by the user with at least a privilege level of user, where the user is defined in the *Job_Owner* attribute (USER).

n Indicates that none of the *Hold_Types* attributes are set (NO_HOLD).

An implementation may define other hold types. Any additional hold types, how they are specified, their internal representation, their behavior, and how they affect the behavior of other utilities are implementation-defined.

The value of the *Hold_Types* attribute shall be the union of the valid hold types (`'s'`, `'o'`, `'u'`, and any implementation-defined hold types), or `'n'`.

A batch server shall not execute a batch job if the *Hold_Types* attribute of the batch job has a value other than NO_HOLD. If the *Hold_Types* attribute of the batch job has a value other than NO_HOLD, the batch job shall be in the HELD state.

Job_Owner Attribute

The *Job_Owner* attribute consists of a pair of user name and host name values of the form:

```
username@hostname
```

A batch server that accepts a *Queue Batch Job Request* shall set the *Job_Owner* attribute to a string that is the *username@hostname* of the user who submitted the job.

Join_Path Attribute

A batch server that executes a batch job for which the value of the *Join_Path* attribute is TRUE shall ignore the value of the *Error_Path* attribute and merge the standard error of the batch job with the standard output of the batch job.

Keep_Files Attribute

A batch server that executes a batch job for which the value of the *Keep_Files* attribute includes the value KEEP_STD_OUTPUT shall retain the standard output of the batch job on the host where execution occurs. The standard output shall be retained in the home directory of the user under whose user ID the batch job is executed and the filename shall be the default filename for the standard output as defined under the **−o** option of the *qsub* utility. The *Output_Path* attribute is not modified.

A batch server that executes a batch job for which the value of the *Keep_Files* attribute includes the value KEEP_STD_ERROR shall retain the standard error of the batch job on the host where execution occurs. The standard error shall be retained in the home directory of the user under whose user ID the batch job is executed and the filename shall be the default filename for

standard error as defined under the **–e** option of the *qsub* utility. The *Error_Path* attribute is not modified.

A batch server that executes a batch job for which the value of the *Keep_Files* attribute includes values other than KEEP_STD_OUTPUT and KEEP_STD_ERROR shall retain these other files on the host where execution occurs. These files (with implementation-defined names) shall be retained in the home directory of the user under whose user identifier the batch job is executed.

Mail_Points and Mail_Users Attributes

A batch server that executes a batch job for which one of the values of the *Mail_Points* attribute is the value MAIL_AT_BEGINNING shall send a mail message to each user account listed in the *Mail_Users* attribute of the batch job.

The mail message shall contain at least the batch job identifier, queue, and server at which the batch job currently resides, and the *Job_Owner* attribute.

Output_Path Attribute

The *Output_Path* attribute of a running job cannot be changed by a *Modify Batch Job Request*. When the *Keep_Files* attribute of the batch job does not contain the value KEEP_STD_OUTPUT, a batch server that executes a batch job shall either:

- Set the standard output stream of the session leader of the batch job to the destination described by the value of the *Output_Path* attribute of the batch job.

  or:

- Buffer the standard output of the session leader of the batch job until completion of the batch job, and when the batch job exits return the contents to the destination described by the value of the *Output_Path* attribute of the batch job.

When the *Output_Path* attribute does not specify a host name, then the batch server shall retain the standard output of the batch job on the host of execution.

When the *Keep_Files* attribute does not contain the value KEEP_STD_OUTPUT and the *Output_Path* attribute does specify a host name, then the final destination of the standard output of the batch job shall be on the host specified.

If the path specified in the *Output_Path* attribute of the batch job is a relative path, the batch server shall expand the path relative to the home directory of the user on the host to which the file is being returned.

Whether or not the batch server buffers the standard output of the batch job until completion of the batch job is implementation-defined. Applications shall not rely on having access to the standard output of a batch job prior to the completion of the batch job.

When the batch server does buffer the standard output of the batch job and the file cannot be opened for write upon completion of the batch job, then the batch server shall place the standard output in an implementation-defined location and notify the user of the location via mail. It shall be possible for the user to process this mail using the *mailx* utility.

If a batch server that does not buffer the standard output cannot open the standard output path of the batch job for write access, then the batch server shall abort the batch job.

4202 **Priority Attribute**

4203 A batch server implementation may choose to preferentially execute a batch job based on the
4204 *Priority* attribute. The interpretation of the batch job *Priority* attribute by a batch server is
4205 implementation-defined. If an implementation uses the *Priority* attribute, it shall interpret larger
4206 values of the *Priority* attribute to mean the batch job shall be preferentially selected for execution.

4207 **Rerunable Attribute**

4208 A batch job that began execution but did not complete, because the batch server either shut
4209 down or terminated abnormally, shall be requeued if the *Rerunable* attribute of the batch job has
4210 the value TRUE.

4211 If a batch job, which was requeued after beginning execution but prior to completion, has a valid
4212 checkpoint file and the batch server supports checkpointing, then the batch job shall be restarted
4213 from the last valid checkpoint.

4214 If the batch job cannot be restarted from a checkpoint, then when a batch job has a *Rerunable*
4215 attribute value of TRUE and was requeued after beginning execution but prior to completion,
4216 the batch server shall place the batch job into execution at the beginning of the job.

4217 When a batch job has a *Rerunable* attribute value other than TRUE and was requeued after
4218 beginning execution but prior to completion, and the batch job cannot be restarted from a
4219 checkpoint, then the batch server shall abort the batch job.

4220 **Resource_List Attribute**

4221 A batch server that executes a batch job shall establish the resource limits of the session leader of
4222 the batch job according to the values of the *Resource_List* attribute of the batch job. Resource
4223 limits shall be enforced by an implementation-defined method.

4224 **Shell_Path_List Attribute**

4225 The *Shell_Path_List* job attribute consists of a list of pairs of pathname and host name values. The
4226 host name component can be omitted, in which case the pathname serves as the default
4227 pathname when a batch server cannot find the name of the host on which it is running in the list.

4228 A batch server that executes a batch job shall select, from the value of the *Shell_Path_List*
4229 attribute of the batch job, a pathname where the shell to execute the batch job shall be found. The
4230 batch server shall select the pathname, in order of preference, according to the following
4231 methods:

4232 • Select the pathname that contains the name of the host on which the batch server is running.

4233 • Select the pathname for which the host name has been omitted.

4234 • Select the pathname for the login shell of the user under which the batch job is to execute.

4235 If the shell path value selected is an invalid pathname, the batch server shall abort the batch job.

4236 If the value of the selected pathname from the *Shell_Path_List* attribute of the batch job
4237 represents a partial path, the batch server shall expand the path relative to a path that is
4238 implementation-defined.

4239 The batch server that executes the batch job shall execute the program that was selected from the
4240 *Shell_Path_List* attribute of the batch job. The batch server shall pass the path to the script of the
4241 batch job as the first argument to the shell program.

User_List Attribute

The *User_List* job attribute consists of a list of pairs of user name and host name values. The host name component can be omitted, in which case the user name serves as a default when a batch server cannot find the name of the host on which it is running in the list.

A batch server that executes a batch job shall select, from the value of the *User_List* attribute of the batch job, a user name under which to create the session leader. The server shall select the user name, in order of preference, according to the following methods:

- Select the user name of a value that contains the name of the host on which the batch server executes.
- Select the user name of a value for which the host name has been omitted.
- Select the user name from the *Job_Owner* attribute of the batch job.

Variable_List Attribute

A batch server that executes a batch job shall create, in the environment of the session leader of the batch job, each environment variable listed in the *Variable_List* attribute of the batch job, and set the value of each such environment variable to that of the corresponding variable in the variable list.

3.2.2.2 Batch Job Routing

To route a batch job is to select a queue from a list and move the batch job to that queue.

A batch server that has routing queues, which have been started, shall route the jobs in the routing queues owned by the batch server. A batch server may delay the routing of a batch job. The algorithm for selecting a batch job and the queue to which it will be routed is implementation-defined.

When a routing queue has multiple possible destinations specified, then the precedence of the destinations is implementation-defined.

A batch server that routes a batch job to a queue at another server shall move the batch job into the target queue with a *Queue Batch Job Request*.

If the target server rejects the *Queue Batch Job Request*, the routing server shall retry routing the batch job or abort the batch job. A batch server that retries failed routings shall provide a means for the batch administrator to specify the number of retries and the minimum period of time between retries. The means by which an administrator specifies the number of retries and the delay between retries is implementation-defined. When the number of retries specified by the batch administrator has been exhausted, the batch server shall abort the batch job and perform the functions of *Batch Job Exit*; see Section 3.2.2.3.

3.2.2.3 Batch Job Exit

For each job in the EXITING state, the batch server that exited the batch job shall perform the following deferred services in the order specified:

1. If buffering standard error, move that file into the location specified by the *Error_Path* attribute of the batch job.
2. If buffering standard output, move that file into the location specified by the *Output_Path* attribute of the batch job.
3. If the *Mail_Points* attribute of the batch job includes MAIL_AT_EXIT, send mail to the users listed in the *Mail_Users* attribute of the batch job. The mail message shall contain at least

the batch job identifier, queue, and server at which the batch job currently resides, and the *Job_Owner* attribute.

4. Remove the batch job from the queue.

If a batch server that buffers the standard error output cannot return the standard error file to the standard error path at the time the batch job exits, the batch server shall do one of the following:

- Mail the standard error file to the batch job owner.
- Save the standard error file and mail the location and name of the file where the standard error is stored to the batch job owner.
- Save the standard error file and notify the user by other implementation-defined means.

If a batch server that buffers the standard output cannot return the standard output file to the standard output path at the time the batch job exits, the batch server shall do one of the following:

- Mail the standard output file to the batch job owner.
- Save the standard output file and mail the location and name of the file where the standard output is stored to the batch job owner.
- Save the standard output file and notify the user by other implementation-defined means.

At the conclusion of job exit processing, the batch job is no longer managed by a batch server.

3.2.2.4 Batch Server Restart

A batch server that has been either shutdown or terminated abnormally, and has returned to operation, is said to have *restarted*.

Upon restarting, a batch server shall requeue those jobs managed by the batch server that were in the RUNNING state at the time the batch server shut down and for which the *Rerunable* attribute of the batch job has the value TRUE.

Queues are defined to be non-volatile. A batch server shall store the content of queues that it controls in such a way that server and system shutdowns do not erase the content of the queues.

3.2.2.5 Batch Job Abort

A batch server that cannot perform a deferred service for a batch job shall abort the batch job.

A batch server that aborts a batch job shall perform the following services:

- Delete the batch job from the queue in which it resides.
- If the *Mail_Points* attribute of the batch job includes the value MAIL_AT_ABORT, send mail to the users listed in the value of the *Mail_Users* attribute of the job. The mail message shall contain at least the batch job identifier, queue, and server at which the batch job currently resides, the *Job_Owner* attribute, and the reason for the abort.
- If the batch job was in the RUNNING state, terminate the session leader of the executing job by sending the session leader a SIGKILL, place the batch job in the EXITING state, and perform the actions of *Batch Job Exit*.

3.2.3 Requested Batch Services

This section describes the services provided by batch servers in response to requests from clients. Table 3-5 summarizes the current set of batch service requests and for each gives its type (deferred or not) and whether it is an optional function.

Table 3-5 Batch Services Summary

Batch Service	Deferred	Optional
Batch Job Execution	Yes	No
Batch Job Routing	Yes	No
Batch Job Exit	Yes	No
Batch Server Restart	Yes	No
Batch Job Abort	Yes	No
Delete Batch Job Request	No	No
Hold Batch Job Request	No	No
Batch Job Message Request	No	Yes
Batch Job Status Request	No	No
Locate Batch Job Request	No	Yes
Modify Batch Job Request	No	No
Move Batch Job Request	No	No
Queue Batch Job Request	No	No
Batch Queue Status Request	No	No
Release Batch Job Request	No	No
Rerun Batch Job Request	No	No
Select Batch Jobs Request	No	No
Server Shutdown Request	No	No
Server Status Request	No	No
Signal Batch Job Request	No	No
Track Batch Job Request	No	Yes

If a request is rejected because the batch client is not authorized to perform the action, the batch server shall return the same status as when the batch job does not exist.

3.2.3.1 Delete Batch Job Request

A batch job is defined to have been deleted when it has been removed from the queue in which it resides and not instantiated in another queue. A client requests that the server that manages a batch job delete the batch job. Such a request is called a *Delete Batch Job Request*.

A batch server shall reject a *Delete Batch Job Request* if any of the following statements are true:

- The user of the batch client is not authorized to delete the designated job.
- The designated job is not managed by the batch server.
- The designated job is in a state inconsistent with the delete request.

A batch server may reject a *Delete Batch Job Request* for other implementation-defined reasons. The method used to determine whether the user of a client is authorized to perform the requested action is implementation-defined.

A batch server requested to delete a batch job shall delete the batch job if the batch job exists and is not in the EXITING state.

A batch server that deletes a batch job in the RUNNING state shall send a SIGKILL signal to the session leader of the batch job. It is implementation-defined whether additional signals are sent

to the session leader of the job prior to sending the SIGKILL signal.

A batch server that deletes a batch job in the RUNNING state shall place the batch job in the EXITING state after it has killed the session leader of the batch job and shall perform the actions of *Batch Job Exit*.

3.2.3.2 *Hold Batch Job Request*

A batch client can request that the batch server add one or more holds to a batch job. Such a request is called a *Hold Batch Job Request*.

A batch server shall reject a *Hold Batch Job Request* if any of the following statements are true:

- The batch server does not support one or more of the requested holds to be added to the batch job.
- The user of the batch client is not authorized to add one or more of the requested holds to the batch job.
- The batch server does not manage the specified job.
- The designated job is in the EXITING state.

A batch server may reject a *Hold Batch Job Request* for other implementation-defined reasons. The method used to determine whether the user of a client is authorized to perform the requested action is implementation-defined.

A batch server that accepts a *Hold Batch Job Request* for a batch job in the RUNNING state shall place a hold on the batch job. The effects, if any, the hold will have on a batch job in the RUNNING state are implementation-defined.

A batch server that accepts a *Hold Batch Job Request* shall add each type of hold listed in the *Hold Batch Job Request*, that is not already present, to the value of the *Hold_Types* attribute of the batch job.

3.2.3.3 *Batch Job Message Request*

Batch Job Message Request is an optional feature of batch servers. If an implementation supports *Batch Job Message Request*, the statements in this section apply and the configuration variable POSIX2_PBS_MESSAGE shall be set to 1.

A batch client can request that a batch server write a message into certain output files of a batch job. Such a request is called a *Batch Job Message Request*.

A batch server shall reject a *Batch Job Message Request* if any of the following statements are true:

- The batch server does not support sending messages to jobs.
- The user of the batch client is not authorized to post a message to the designated job.
- The designated job does not exist on the batch server.
- The designated job is not in the RUNNING state.

A batch server may reject a *Batch Job Message Request* for other implementation-defined reasons. The method used to determine whether the user of a client is authorized to perform the requested action is implementation-defined.

A batch server that accepts a *Batch Job Message Request* shall write the message sent by the batch client into the files indicated by the batch client.

3.2.3.4 *Batch Job Status Request*

A batch client can request that a batch server respond with the status and attributes of a batch job. Such a request is called a *Batch Job Status Request*.

A batch server shall reject a *Batch Job Status Request* if any of the following statements are true:

- The user of the batch client is not authorized to query the status of the designated job.
- The designated job is not managed by the batch server.

A batch server may reject a *Batch Job Status Request* for other implementation-defined reasons. The method used to determine whether the user of a client is authorized to perform the requested action is implementation-defined.

A batch server that accepts a *Batch Job Status Request* shall return a *Batch Job Status Message* to the batch client.

A batch server may return other information in response to a *Batch Job Status Request*.

3.2.3.5 *Locate Batch Job Request*

Locate Batch Job Request is an optional feature of batch servers. If an implementation supports *Locate Batch Job Request*, the statements in this section apply and the configuration variable POSIX2_PBS_LOCATE shall be set to 1.

A batch client can ask a batch server to respond with the location of a batch job that was created by the batch server. Such a request is called a *Locate Batch Job Request*.

A batch server that accepts a *Locate Batch Job Request* shall return a *Batch Job Location Message* to the batch client.

A batch server may reject a *Locate Batch Job Request* for a batch job that was not created by that server.

A batch server may reject a *Locate Batch Job Request* for a batch job that is no longer managed by that server; that is, for a batch job that is not in a queue owned by that server.

A batch server may reject a *Locate Batch Job Request* for other implementation-defined reasons.

3.2.3.6 *Modify Batch Job Request*

Batch clients modify (alter) the attributes of a batch job by making a request to the server that manages the batch job. Such a request is called a *Modify Batch Job Request*.

A batch server shall reject a *Modify Batch Job Request* if any of the following statements are true:

- The user of the batch client is not authorized to make the requested modification to the batch job.
- The designated job is not managed by the batch server.
- The requested modification is inconsistent with the state of the batch job.
- An unrecognized resource is requested for a batch job in an execution queue.

A batch server may reject a *Modify Batch Job Request* for other implementation-defined reasons. The method used to determine whether the user of a client is authorized to perform the requested action is implementation-defined.

A batch server that accepts a *Modify Batch Job Request* shall modify all the specified attributes of the batch job. A batch server that rejects a *Modify Batch Job Request* shall modify none of the attributes of the batch job.

If the servicing by a batch server of an otherwise valid request would result in no change, then the batch server shall indicate successful completion of the request.

3.2.3.7 Move Batch Job Request

A batch client can request that a batch server move a batch job to another destination. Such a request is called a *Move Batch Job Request*.

A batch server shall reject a *Move Batch Job Request* if any of the following statements are true:

- The user of the batch client is not authorized to remove the designated job from the queue in which the batch job resides.
- The user of the batch client is not authorized to move the designated job to the destination.
- The designated job is not managed by the batch server.
- The designated job is in the EXITING state.
- The destination is inaccessible.

A batch server can reject a *Move Batch Job Request* for other implementation-defined reasons. The method used to determine whether the user of a client is authorized to perform the requested action is implementation-defined.

A batch server that accepts a *Move Batch Job Request* shall perform the following services:

- Queue the designated job at the destination.
- Remove the designated job from the queue in which the batch job resides.

If the destination resides on another batch server, the batch server shall queue the batch job at the destination by sending a *Queue Batch Job Request* to the other server. If the *Queue Batch Job Request* fails, the batch server shall reject the *Move Batch Job Request*. If the *Queue Batch Job Request* succeeds, the batch server shall remove the batch job from its queue.

The batch server shall not modify any attributes of the batch job.

3.2.3.8 Queue Batch Job Request

A batch queue is controlled by one and only one batch server. A batch server is said to own the queues that it controls. Batch clients make requests of batch servers to have jobs queued. Such a request is called a *Queue Batch Job Request*.

A batch server requested to queue a batch job for which the queue is not specified shall select an implementation-defined queue for the batch job. Such a queue is called the *default queue* of the batch server. The implementation shall provide the means for a batch administrator to specify the default queue. The queue, whether specified or defaulted, is called the *target queue.*

A batch server shall reject a *Queue Batch Job Request* if any of the following statements are true:

- The client is not authorized to create a batch job in the target queue.
- The request specifies a queue that does not exist on the batch server.
- The target queue is an execution queue and the batch server cannot satisfy a resource requirement of the batch job.
- The target queue is an execution queue and an unrecognized resource is requested.
- The target queue is an execution queue, the batch server does not support checkpointing, and the value of the *Checkpoint* attribute of the batch job is not NO_CHECKPOINT.

- The job requires access to a user identifier that the batch client is not authorized to access.

A batch server may reject a *Queue Batch Job Request* for other implementation-defined reasons.

A batch server that accepts a *Queue Batch Job Request* for a batch job for which the PBS_O_QUEUE value is missing from the value of the *Variable_List* attribute of the batch job shall add that variable to the list and set the value to the name of the target queue. Once set, no server shall change the value of PBS_O_QUEUE, even if the batch job is moved to another queue.

A batch server that accepts a *Queue Batch Job Request* for a batch job for which the PBS_JOBID value is missing from the value of the *Variable_List* attribute shall add that variable to the list and set the value to the batch job identifier assigned by the server in the format:

```
sequence_number.server
```

A batch server that accepts a *Queue Batch Job Request* for a batch job for which the PBS_JOBNAME value is missing from the value of the *Variable_List* attribute of the batch job shall add that variable to the list and set the value to the *Job_Name* attribute of the batch job.

3.2.3.9 Batch Queue Status Request

A batch client can request that a batch server respond with the status and attributes of a queue. Such a request is called a *Batch Queue Status Request*.

A batch server shall reject a *Batch Queue Status Request* if any of the following statements are true:

- The user of the batch client is not authorized to query the status of the designated queue.
- The designated queue does not exist on the batch server.

A batch server may reject a *Batch Queue Status Request* for other implementation-defined reasons. The method used to determine whether the user of a client is authorized to perform the requested action is implementation-defined.

A batch server that accepts a *Batch Queue Status Request* shall return a *Batch Queue Status Reply* to the batch client.

3.2.3.10 Release Batch Job Request

A batch client can request that server remove one or more holds from a batch job. Such a request is called a *Release Batch Job Request*.

A batch server shall reject a *Release Batch Job Request* if any of the following statements are true:

- The user of the batch client is not authorized to remove one or more of the requested holds from the batch job.
- The batch server does not manage the specified job.

A batch server may reject a *Release Batch Job Request* for other implementation-defined reasons. The method used to determine whether the user of a client is authorized to perform the requested action is implementation-defined.

A batch server that accepts a *Release Batch Job Request* shall remove each type of hold listed in the *Release Batch Job Request*, that is present, from the value of the *Hold_Types* attribute of the batch job.

4521 *3.2.3.11 Rerun Batch Job Request*

4522 To rerun a batch job is to kill the session leader of the batch job and leave the batch job eligible
4523 for re-execution. A batch client can request that a batch server rerun a batch job. Such a request is
4524 called *Rerun Batch Job Request*.

4525 A batch server shall reject a *Rerun Batch Job Request* if any of the following statements are true:

4526 • The user of the batch client is not authorized to rerun the designated job.

4527 • The *Rerunable* attribute of the designated job has the value FALSE.

4528 • The designated job is not in the RUNNING state.

4529 • The batch server does not manage the designated job.

4530 A batch server may reject a *Rerun Batch Job Request* for other implementation-defined reasons. |
4531 The method used to determine whether the user of a client is authorized to perform the |
4532 requested action is implementation-defined. |

4533 A batch server that rejects a *Rerun Batch Job Request* shall in no way modify the execution of the
4534 batch job.

4535 A batch server that accepts a request to rerun a batch job shall perform the following services:

4536 • Requeue the batch job in the execution queue in which it was executing.

4537 • Send a SIGKILL signal to the process group of the session leader of the batch job.

4538 An implementation may indicate to the batch job owner that the batch job has been rerun. |
4539 Whether and how the batch job owner is notified that a batch job is rerun is implementation- |
4540 defined. |

4541 A batch server that reruns a batch job may send other implementation-defined signals to the |
4542 session leader of the batch job prior to sending the SIGKILL signal. |

4543 A batch server may preferentially select a rerun job for execution. Whether rerun jobs shall be |
4544 selected for execution before other jobs is implementation-defined. |

4545 *3.2.3.12 Select Batch Jobs Request*

4546 A batch client can request from a batch server a list of jobs managed by that server that match a
4547 list of selection criteria. Such a request is called a *Select Batch Jobs Request*. All the batch jobs
4548 managed by the batch server that receives the request are candidates for selection.

4549 A batch server that accepts a *Select Batch Jobs Request* shall return a list of zero or more job
4550 identifiers that correspond to jobs that meet the selection criteria.

4551 If the batch client is not authorized to query the status of a batch job, the batch server shall not
4552 select the batch job.

4553 *3.2.3.13 Server Shutdown Request*

4554 A batch server is defined to have shut down when it does not respond to requests from clients
4555 and does not perform deferred services for jobs. A batch client can request that a batch server
4556 shut down. Such a request is called a *Server Shutdown Request*.

4557 A batch server shall reject a *Server Shutdown Request* from a client that is not authorized to shut |
4558 down the batch server. The method used to determine whether the user of a client is authorized |
4559 to perform the requested action is implementation-defined. |

A batch server may reject a *Server Shutdown Request* for other implementation-defined reasons. The reasons for which a *Server Shutdown Request* may be rejected are implementation-defined.

At server shutdown, a batch server shall do, in order of preference, one of the following:

- If checkpointing is implemented and the batch job is checkpointable, then checkpoint the batch job and requeue it.
- If the batch job is rerunable, then requeue the batch job to be rerun (restarted from the beginning).
- Abort the batch job.

3.2.3.14 Server Status Request

A batch client can request that a batch server respond with the status and attributes of the batch server. Such a request is called a *Server Status Request*.

A batch server shall reject a *Server Status Request* if the following statement is true:

- The user of the batch client is not authorized to query the status of the designated server.

A batch server may reject a *Server Status Request* for other implementation-defined reasons. The method used to determine whether the user of a client is authorized to perform the requested action is implementation-defined.

A batch server that accepts a *Server Status Request* shall return a *Server Status Reply* to the batch client.

3.2.3.15 Signal Batch Job Request

A batch client can request that a batch server signal the session leader of a batch job. Such a request is called a *Signal Batch Job Request*.

A batch server shall reject a *Signal Batch Job Request* if any of the following statements are true:

- The user of the batch client is not authorized to signal the batch job.
- The job is not in the RUNNING state.
- The batch server does not manage the designated job.
- The requested signal is not supported by the implementation.

A batch server may reject a *Signal Batch Job Request* for other implementation-defined reasons. The method used to determine whether the user of a client is authorized to perform the requested action is implementation-defined.

A batch server that accepts a request to signal a batch job shall send the signal requested by the batch client to the process group of the session leader of the batch job.

3.2.3.16 Track Batch Job Request

Track Batch Job Request is an optional feature of batch servers. If an implementation supports *Track Batch Job Request*, the statements in this section apply and the configuration variable POSIX2_PBS_TRACK shall be set to 1.

Track Batch Job Request provides a method for tracking the current location of a batch job. Clients may use the tracking information to determine the batch server that should receive a batch server request.

If *Track Batch Job Request* is supported by a batch server, then when the batch server queues a batch job as a result of a *Queue Batch Job Request*, and the batch server is not the batch server that created the batch job, the batch server shall send a *Track Batch Job Request* to the batch server that created the job.

If *Track Batch Job Request* is supported by a batch server, then the *Track Batch Job Request* may also be sent to other servers as a backup to the primary server. The method by which backup servers are specified is implementation-defined.

If *Track Batch Job Request* is supported by a batch server that receives a *Track Batch Job Request*, then the batch server shall record the current location of the batch job as contained in the request.

3.3 Common Behavior for Batch Environment Utilities

3.3.1 Batch Job Identifier

A utility shall recognize *job_identifier*s of the format:

```
[sequence_number][.server_name][@server]
```

where:

sequence_number An integer that, when combined with *server_name*, provides a batch job identifier that is unique within the batch system.

server_name The name of the batch server to which the batch job was originally submitted.

server The name of the batch server that is currently managing the batch job.

If the application omits the batch *server_name* portion of a batch job identifier, a utility shall use the name of a default batch server.

If the application omits the batch *server* portion of a batch job identifier, a utility shall use:

- The batch server indicated by *server_name*, if present.
- The name of the default batch server.
- The name of the batch server that is currently managing the batch job.

If only *@server* is specified, then the status of all jobs owned by the user on the requested server is listed.

The means by which a utility determines the default batch server is implementation-defined.

If the application presents the batch *server* portion of a batch job identifier to a utility, the utility shall send the request to the specified server.

A strictly conforming application shall use the syntax described for the job identifier. Whenever a batch job identifier is specified whose syntax is not recognized by an implementation, then a message for each error that occurs shall be written to standard error and the utility shall exit with an exit status greater than zero.

When a batch job identifier is supplied as an argument to a batch utility and the *server_name* portion of the batch job identifier is omitted, then the utility shall use the name of the default batch server.

When a batch job identifier is supplied as an argument to a batch utility and the batch *server* portion of the batch job identifier is omitted, then the utility shall use either:

- The name of the default batch server

or:

- The name of the batch server that is currently managing the batch job

When a batch job identifier is supplied as an argument to a batch utility and the batch *server* portion of the batch job identifier is specified, then the utility shall send the required *Batch Server Request* to the specified server.

3.3.2 Destination

The utility shall recognize a *destination* of the format:

```
[queue][@server]
```

where:

queue The name of a valid execution or routing queue at the batch server denoted by *@server*, defined as a string of up to 15 alphanumeric characters in the portable character set (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set) where the first character is alphabetic.

server The name of a batch server, defined as a string of alphanumeric characters in the portable character set.

If the application omits the batch *server* portion of a destination, then the utility shall use either:

- The name of the default batch server

or:

- The name of the batch server that is currently managing the batch job

The means by which a utility determines the default batch server is implementation-defined.

If the application omits the *queue* portion of a destination, then the utility shall use the name of the default queue at the batch server chosen. The means by which a batch server determines its default queue is implementation-defined. If a destination is specified in the *queue@server* form, then the utility shall use the specified queue at the specified server.

A strictly conforming application shall use the syntax described for a destination. Whenever a destination is specified whose syntax is not recognized by an implementation, then a message shall be written to standard error and the utility shall exit with an exit status greater than zero.

3.3.3 Multiple Keyword-Value Pairs

For each option that can have multiple keyword-value pair arguments, the following rules shall apply. Examples of options that can have list-oriented option-arguments are **–u** *value@keyword* and **–l** *keyword=value*.

1. If a batch utility is presented with a list-oriented option-argument for which a keyword has a corresponding value that begins with a single or double quote, then the utility shall stop interpreting the input stream for delimiters until a second single or double quote, respectively, is encountered. This feature allows some flexibility for a comma (`','`) or equals sign (`'='`) to be part of the value string for a particular keyword; for example:

   ```
   keywd1='val1,val2',keywd2="val3,val4"
   ```

   Note: This may require the user to escape the quotes as in the following command:

```
foo –xkeywd1=\'val1,val2\',keywd2=\"val3,val4\"
```

2. If a batch server is presented with a list-oriented attribute that has a keyword that was encountered earlier in the list, then the later entry for that keyword shall replace the earlier entry.

3. If a batch server is presented with a list-oriented attribute that has a keyword without any corresponding value of the form *keyword*= or *@keyword* and the same keyword was encountered earlier in the list, then the prior entry for that keyword shall be ignored by the batch server.

4. If a batch utility is expecting a list-oriented option-argument entry of the form *keyword=value*, but is presented with an entry of the form *keyword* without any corresponding *value*, then the entry shall be treated as though a default value of NULL was assigned (that is, *keyword*=NULL) for entry parsing purposes. The utility shall include only the keyword, not the NULL value, in the associated job attribute.

5. If a batch utility is expecting a list-oriented option-argument entry of the form *value@keyword*, but is presented with an entry of the form *value* without any corresponding *keyword*, then the entry shall be treated as though a keyword of NULL was assigned (that is, *value*@NULL) for entry parsing purposes. The utility shall include only the value, not the NULL keyword, in the associated job attribute.

6. A batch server shall accept a list-oriented attribute that has multiple occurrences of the same keyword, interpreting the keywords, in order, with the last value encountered taking precedence over prior instances of the same keyword. This rule allows, but does not require, a batch utility to preprocess the attribute to remove duplicate keywords.

7. If a batch utility is presented with multiple list-oriented option-arguments on the command line or in script directives, or both, for a single option, then the utility shall concatenate, in order, any command line keyword and value pairs to the end of any directive keyword and value pairs separated by a single comma to produce a single string that is an equivalent, valid option-argument. The resulting string shall be assigned to the associated attribute of the batch job (after optionally removing duplicate entries as described in item 6.

Chapter 4

Utilities

This chapter contains the definitions of the utilities, as follows:

- Mandatory utilities that are present on every conformant system
- Optional utilities that are present only on systems supporting the associated option; see Section 1.8.1 (on page 2210) for information on the options in this volume of IEEE Std 1003.1-200x

NAME

admin — create and administer SCCS files (**DEVELOPMENT**)

SYNOPSIS

XSI

```
admin –i[name][–n][–a login][–d flag][–e login][–f flag][–m mrlist]
    [–r rel][–t[name][–y[comment]] newfile
admin –n[–a login][–d flag][–e login][–f flag][–m mrlist][–t[name]]
    [–y[comment]] newfile ...
admin [–a login][–d flag][–m mrlist][–r rel][–t[name]] file ...
admin –h file ...
admin –z file ...
```

DESCRIPTION

The *admin* utility shall create new SCCS files or change parameters of existing ones. If a named file does not exist, it shall be created, and its parameters shall be initialized according to the specified options. Parameters not initialized by an option shall be assigned a default value. If a named file does exist, parameters corresponding to specified options shall be changed, and other parameters shall be left as is.

All SCCS filenames supplied by the application shall be of the form s.*filename*. New SCCS files shall be given read-only permission mode. Write permission in the parent directory is required to create a file. All writing done by *admin* shall be to a temporary *x-file*, named x.*filename* (see *get*) created with read-only mode if *admin* is creating a new SCCS file, or created with the same mode as that of the SCCS file if the file already exists. After successful execution of *admin*, the SCCS file shall be removed (if it exists), and the *x-file* shall be renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occur.

The *admin* utility shall also use a transient lock file (named z.*filename*), which is used to prevent simultaneous updates to the SCCS file; see *get* (on page 2675).

OPTIONS

The *admin* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines, except that the **–i**, **–t**, and **–y** options have optional option-arguments. These optional option-arguments shall not be presented as separate arguments. The following options are supported:

–n Create a new SCCS file. When **–n** is used without **–i**, the SCCS file shall be created with control information but without any file data.

–i[*name***]** Specify the *name* of a file from which the text for a new SCCS file shall be taken. The text constitutes the first delta of the file (see the **–r** option for delta numbering scheme). If the **–i** option is used, but the *name* option-argument is omitted, the text shall be obtained by reading the standard input. If this option is omitted, the SCCS file shall be created with control information but without any file data. The **–i** option implies the **–n** option.

–r *SID* Specify the SID of the initial delta to be inserted. This SID shall be a trunk SID; that is, the branch and sequence numbers shall be zero or missing. The level number is optional, and defaults to 1.

–t[*name***]** Specify the *name* of a file from which descriptive text for the SCCS file shall be taken. In the case of existing SCCS files (neither **–i** nor **–n** is specified):

- A **–t** option without a *name* option-argument shall cause the removal of descriptive text (if any) currently in the SCCS file.
- A **–t** option with a *name* option-argument shall cause the text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.

–f *flag* Specify a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. Several **–f** options may be supplied on a single *admin* command line. Implementations shall recognize the following flags and associated values:

b Allow use of the **–b** option on a *get* command to create branch deltas.

c*ceil* Specify the highest release (that is, ceiling), a number less than or equal to 9 999, which may be retrieved by a *get* command for editing. The default value for an unspecified **c** flag shall be 9 999.

f*floor* Specify the lowest release (that is, floor), a number greater than 0 but less than 9 999, which may be retrieved by a *get* command for editing. The default value for an unspecified **f** flag shall be 1.

d*SID* Specify the default delta number (SID) to be used by a *get* command.

i*str* Treat the ''No ID keywords'' message issued by *get* or *delta* as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords (see *get* (on page 2675)) are found in the text retrieved or stored in the SCCS file. If a value is supplied, the application shall ensure that the keywords exactly match the given string; however, the string shall contain a keyword, and no embedded <newline>s.

j Allow concurrent *get* commands for editing on the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.

l*list* Specify a *list* of releases to which deltas can no longer be made (that is, *get* **–e** against one of these locked releases fails). Conforming applications shall use the following syntax to specify a *list*. Implementations may accept additional forms as an extension:

```
<list> ::= a | <range-list>
<range-list> ::= <range> | <range-list>, <range>
<range> ::= <SID>
```

The character *a* in the *list* shall be equivalent to specifying all releases for the named SCCS file. The non-terminal <*SID*> in range shall be the delta number of an existing delta associated with the SCCS file.

n Cause *delta* to create a null delta in each of those releases (if any) being skipped when a delta is made in a new release (for example, in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas shall serve as anchor points so that branch deltas may later be created from them. The absence of this flag shall cause skipped releases to be nonexistent in the SCCS file, preventing branch deltas from being created from them in the future. During the initial creation of an SCCS file, the **n** flag may be ignored; that is, if the **–r** option is used to set the release number of the initial SID to a value greater than 1, null deltas need not be created for the ''skipped'' releases.

q*text* Substitute user-definable *text* for all occurrences of the %**Q**% keyword in the SCCS file text retrieved by *get*.

m*mod* Specify the module name of the SCCS file substituted for all occurrences of the %**M**% keyword in the SCCS file text retrieved by *get*. If the **m** flag is not specified, the value assigned shall be the name of the SCCS file with the leading `'.'` removed.

t*type* Specify the *type* of module in the SCCS file substituted for all occurrences of the %**Y**% keyword in the SCCS file text retrieved by *get*.

v*pgm* Cause *delta* to prompt for modification request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validation program. (If this flag is set when creating an SCCS file, the application shall ensure that the **m** option is also used even if its value is null.)

–d *flag* Remove (delete) the specified *flag* from an SCCS file. Several **–d** options may be supplied on a single *admin* command. See the **–f** option for allowable *flag* names. (The **l***list* flag gives a *list* of releases to be unlocked. See the **–f** option for further description of the **l** flag and the syntax of a *list*.)

–a *login* Specify a *login* name, or numerical group ID, to be added to the list of users who may make deltas (changes) to the SCCS file. A group ID shall be equivalent to specifying all *login* names common to that group ID. Several **–a** options may be used on a single *admin* command line. As many *login*s, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If *login* or group ID is preceded by a `'!'`, the users so specified shall be denied permission to make deltas.

–e *login* Specify a *login* name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all *login* names common to that group ID. Several **–e** options may be used on a single *admin* command line.

–y[*comment***]** Insert the *comment* text into the SCCS file as a comment for the initial delta in a manner identical to that of *delta*. In the POSIX locale, omission of the **–y** option shall result in a default comment line being inserted in the form:

`"date and time created %s %s by %s", <`*date*`>, <`*time*`>, <`*login*`>`

where <*date*> is expressed in the format of the *date* utility's `%y/%m/%d` conversion specification, <*time*> in the format of the *date* utility's `%T` conversion specification format, and <*login*> is the login name of the user creating the file.

–m *mrlist* Insert the list of modification request (MR) numbers into the SCCS file as the reason for creating the initial delta in a manner identical to *delta*. The application shall ensure that the **v** flag is set and the MR numbers are validated if the **v** flag has a value (the name of an MR number validation program). A diagnostic message shall be written if the **v** flag is not set or MR validation fails.

–h Check the structure of the SCCS file and compare the newly computed checksum (the sum of all the characters in the SCCS file except those in the first line) with the checksum that is stored in the first line of the SCCS file. If the newly computed checksum does not match the checksum in the SCCS file, a diagnostic message shall be written.

–z Recompute the SCCS file checksum and store it in the first line of the SCCS file (see the **–h** option above). Note that use of this option on a truly corrupted file may prevent future detection of the corruption.

OPERANDS

The following operands shall be supported:

file A pathname of an existing SCCS file or a directory. If *file* is a directory, the *admin* utility shall behave as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with **s.**) and unreadable files shall be silently ignored.

newfile A pathname of an SCCS file to be created.

If exactly one *file* or *newfile* operand appears, and it is '–', the standard input shall be read; each line of the standard input shall be taken to be the name of an SCCS file to be processed. Non-SCCS files and unreadable files shall be silently ignored.

STDIN

The standard input shall be a text file used only if the **–i** is specified without an option-argument or if a *file* or *newfile* operand is specified as '–'. If the first character of any standard input line is <SOH> in the POSIX locale, the results are unspecified.

INPUT FILES

The existing SCCS files shall be text files of an unspecified format.

The application shall ensure that the file named by the **–i** option's *name* option-argument shall be a text file; if the first character of any line in this file is <SOH> in the POSIX locale, the results are unspecified. If this file contains more than 99 999 lines, the number of lines recorded in the header for this file shall be 99 999 for this delta.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *admin*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and the contents of the default **–y** comment.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

Any SCCS files created shall be text files of an unspecified format. During processing of a *file*, a locking *z-file*, as described in *get* (on page 2675), may be created and deleted.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

It is recommended that directories containing SCCS files be writable by the owner only, and that SCCS files themselves be read-only. The mode of the directories should allow only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by SCCS commands.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

delta, *get*, *prs*, *what*

CHANGE HISTORY

First released in Issue 2.

Issue 6

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

The normative text is reworded to emphasize the term ‘‘shall’’ for implementation requirements.

The grammar is updated.

The Open Group Base Resolution bwg 2001-007 is applied, adding new text to the INPUT FILES section warning that the maximum lines recorded in the file is 99 999.

NAME

alias — define or display aliases

SYNOPSIS

UP

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

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 Section 2.3.1 (on page 2234).

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 volume of IEEE Std 1003.1-200x, the alias definition shall not affect the parent process of the current shell nor any utility environment invoked by the shell; see Section 2.12 (on page 2263).

OPTIONS

None.

OPERANDS

The following operands shall be supported:

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.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

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

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

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 for reinput to the shell. See the description of shell quoting in Section 2.2 (on page 2232).

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

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

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

1. Change *ls* to give a columnated, more annotated output:

   ```
   alias ls="ls –CF"
   ```

2. Create a simple ‘‘redo’’ command to repeat previous entries in the command history file:

   ```
   alias r='fc –s'
   ```

3. Use 1K units for *du*:

   ```
   alias du=du\ –k
   ```

4. Set up *nohup* so that it can deal with an argument that is itself an alias name:

   ```
   alias nohup="nohup "
   ```

RATIONALE

The *alias* description is based on 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 this volume of IEEE Std 1003.1-200x, 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 this volume of IEEE Std 1003.1-200x only when an explicit extension such as **–x** is used. The standard developers considered 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 had not written aliases in a quoted manner suitable for reentry to the shell, but this volume of IEEE Std 1003.1-200x has made this a requirement for all similar output. Therefore, consistency with this volume of IEEE Std 1003.1-200x was chosen over this detail of historical practice.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.9.5 (on page 2256)

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The APPLICATION USAGE section is added.

NAME

ar — create and maintain library archives

SYNOPSIS

SD `ar –d[–v] archive file ...`

XSI `ar –m[–abiv][posname] archive file ...`

XSI `ar –p[–v][–s]archive [file ...]`

XSI `ar –q[–cv] archive file ...`

XSI `ar –r[–cuv][–abi][posname]archive file ...`

XSI `ar –t[–v][–s]archive [file ...]`

XSI `ar –x[–v][–sCT]archive [file ...]`

DESCRIPTION

The *ar* utility can be used to create and maintain groups of files combined into an archive. Once an archive has been created, new files can be added, and existing files in an archive can be extracted, deleted, or replaced. When an archive consists entirely of valid object files, the implementation shall format the archive so that it is usable as a library for link editing (see *c99* and *fort77*). When some of the archived files are not valid object files, the suitability of the

XSI archive for library use is undefined. If an archive consists entirely of printable files, the entire archive shall be printable.

When *ar* creates an archive, it creates administrative information indicating whether a symbol table is present in the archive. When there is at least one object file that *ar* recognizes as such in the archive, an archive symbol table shall be created in the archive and maintained by *ar*; it is used by the link editor to search the archive. Whenever the *ar* utility is used to create or update the contents of such an archive, the symbol table shall be rebuilt. The **–s** option shall force the symbol table to be rebuilt.

All *file* operands can be pathnames. However, files within archives shall be named by a filename, which is the last component of the pathname used when the file was entered into the archive. The comparison of *file* operands to the names of files in archives shall be performed by comparing the last component of the operand to the name of the file in the archive.

It is unspecified whether multiple files in the archive may be identically named. In the case of

XSI such files, however, each *file* and *posname* operand shall match only the first file in the archive having a name that is the same as the last component of the operand.

OPTIONS

The *ar* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

XSI **–a** Position new files in the archive after the file named by the *posname* operand.

XSI **–b** Position new files in the archive before the file named by the *posname* operand.

–c Suppress the diagnostic message that is written to standard error by default when the archive *archive* is created.

XSI **–C** Prevent extracted files from replacing like-named files in the file system. This option is useful when **–T** is also used, to prevent truncated filenames from

replacing files with the same prefix.

–d Delete one or more *file*s from *archive*.

XSI **–i** Position new files in the archive before the file in the archive named by the *posname* operand (equivalent to **–b**).

XSI **–m** Move the named files in the archive. The **–a**, **–b**, or **–i** options with the *posname* operand indicate the position; otherwise, move the names files in the archive to the end of the archive.

–p Write the contents of the *file*s in the archive named by *file* operands from *archive* to the standard output. If no *file* operands are specified, the contents of all files in the archive shall be written in the order of the archive.

XSI **–q** Append the named files to the end of the archive. In this case *ar* does not check whether the added files are already in the archive. This is useful to bypass the searching otherwise done when creating a large archive piece by piece.

–r Replace or add *file*s to *archive*. If the archive named by *archive* does not exist, a new archive shall be created and a diagnostic message shall be written to standard error (unless the **–c** option is specified). If no *file*s are specified and the *archive* exists, the results are undefined. Files that replace existing files in the archive shall not change the order of the archive. Files that do not replace existing files in the XSI archive shall be appended to the archive unless a **–a**, **–b**, or **–i** option specifies another position.

XSI **–s** Force the regeneration of the archive symbol table even if *ar* is not invoked with an option that modifies the archive contents. This option is useful to restore the archive symbol table after it has been stripped; see *strip*.

–t Write a table of contents of *archive* to the standard output. The files specified by the *file* operands shall be included in the written list. If no *file* operands are specified, all files in *archive* shall be included in the order of the archive.

XSI **–T** Allow filename truncation of extracted files whose archive names are longer than the file system can support. By default, extracting a file with a name that is too long shall be an error; a diagnostic message shall be written and the file shall not be extracted.

–u Update older files in the archive. When used with the **–r** option, files in the archive shall be replaced only if the corresponding *file* has a modification time that is at least as new as the modification time of the file in the archive.

–v Give verbose output. When used with the option characters **–d**, **–r**, or **–x**, write a detailed file-by-file description of the archive creation and maintenance activity, as described in the STDOUT section.

When used with **–p**, write the name of the file in the archive to the standard output before writing the file in the archive itself to the standard output, as described in the STDOUT section.

When used with **–t**, include a long listing of information about the files in the archive, as described in the STDOUT section.

–x Extract the files in the archive named by the *file* operands from *archive*. The contents of the archive shall not be changed. If no *file* operands are given, all files in the archive shall be extracted. The modification time of each file extracted shall be set to the time the file is extracted from the archive.

5107 **OPERANDS**

5108 The following operands shall be supported:

5109 *archive* A pathname of the archive.

5110 *file* A pathname. Only the last component shall be used when comparing against the
5111 names of files in the archive. If two or more *file* operands have the same last
5112 pathname component (basename), the results are unspecified. The
5113 implementation's archive format shall not truncate valid filenames of files added
5114 to or replaced in the archive.

5115 XSI *posname* The name of a file in the archive, used for relative positioning; see options **–m** and
5116 **–r**.

5117 **STDIN**

5118 Not used.

5119 **INPUT FILES**

5120 The archive named by *archive* shall be a file in the format created by *ar* **–r**.

5121 **ENVIRONMENT VARIABLES**

5122 The following environment variables shall affect the execution of *ar*:

5123 *LANG* Provide a default value for the internationalization variables that are unset or null.
5124 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
5125 Internationalization Variables for the precedence of internationalization variables
5126 used to determine the values of locale categories.)

5127 *LC_ALL* If set to a non-empty string value, override the values of all the other
5128 internationalization variables.

5129 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
5130 characters (for example, single-byte as opposed to multi-byte characters in
5131 arguments and input files).

5132 *LC_MESSAGES*
5133 Determine the locale that should be used to affect the format and contents of
5134 diagnostic messages written to standard error.

5135 *LC_TIME* Determine the format and content for date and time strings written by *ar* **–tv**.

5136 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

5137 *TMPDIR* Determine the pathname that overrides the default directory for temporary files, if
5138 any.

5139 *TZ* Determine the timezone used to calculate date and time strings written by *ar* **–tv**.
5140 If *TZ* is unset or null, an unspecified default timezone shall be used.

5141 **ASYNCHRONOUS EVENTS**

5142 Default.

5143 **STDOUT**

5144 If the **–d** option is used with the **–v** option, the standard output format shall be:

5145 `"d – %s\n", <file>`

5146 where *file* is the operand specified on the command line.

5147 If the **–p** option is used with the **–v** option, *ar* shall precede the contents of each file with:

"\n<%s>\n\n", *<file>*

where *file* is the operand specified on the command line, if *file* operands were specified, and the name of the file in the archive if they were not.

If the **–r** option is used with the **–v** option:

- If *file* is already in the archive, the standard output format shall be:

  "r – %s\n", *<file>*

  where <*file*> is the operand specified on the command line.

- If *file* is not already in the archive, the standard output format shall be:

  "a – %s\n", *<file>*

  where <*file*> is the operand specified on the command line.

If the **–t** option is used, *ar* shall write the names of the files in the archive to the standard output in the format:

"%s\n", *<file>*

where *file* is the operand specified on the command line, if *file* operands were specified, or the name of the file in the archive if they were not.

If the **–t** option is used with the **–v** option, the standard output format shall be:

"%s %u/%u %u %s %d %d:%d %d %s\n", *<member mode>*, *<user ID>*,
<group ID>, *<number of bytes in member>*,
<abbreviated month>, *<day-of-month>*, *<hour>*,
<minute>, *<year>*, *<file>*

where:

<*file*> Shall be the operand specified on the command line, if *file* operands were specified, or the name of the file in the archive if they were not.

<*member mode*>
Shall be formatted the same as the <*file mode*> string defined in the STDOUT section of *ls*, except that the first character, the <*entry type*>, is not used; the string represents the file mode of the file in the archive at the time it was added to or replaced in the archive.

The following represent the last-modification time of a file when it was most recently added to or replaced in the archive:

<*abbreviated month*>
Equivalent to the format of the %b conversion specification format in *date*.

<*day-of-month*>
Equivalent to the format of the %e conversion specification format in *date*.

<*hour*> Equivalent to the format of the %H conversion specification format in *date*.

<*minute*> Equivalent to the format of the %M conversion specification format in *date*.

<*year*> Equivalent to the format of the %Y conversion specification format in *date*.

When *LC_TIME* does not specify the POSIX locale, a different format and order of presentation of these fields relative to each other may be used in a format appropriate in the specified locale.

If the **–x** option is used with the **–v** option, the standard output format shall be:

```
"x – %s\n", <file>
```

where *file* is the operand specified on the command line, if *file* operands were specified, or the name of the file in the archive if they were not.

STDERR

The standard error shall be used only for diagnostic messages. The diagnostic message about creating a new archive when **–c** is not specified shall not modify the exit status.

OUTPUT FILES

Archives are files with unspecified formats.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

The archive format is not described. It is recognized that there are several known *ar* formats, which are not compatible. The *ar* utility is included, however, to allow creation of archives that are intended for use only on one machine. The archive is specified as a file, and it can be moved as a file. This does allow an archive to be moved from one machine to another machine that uses the same implementation of *ar*.

Utilities such as *pax* (and its forebears *tar* and *cpio*) also provide portable ‘‘archives’’. This is a not a duplication; the *ar* utility is included to provide an interface primarily for *make* and the compilers, based on a historical model.

In historical implementations, the **–q** option (available on XSI-conforming systems) is known to execute quickly because *ar* does not check on whether the added members are already in the archive. This is useful to bypass the searching otherwise done when creating a large archive piece-by-piece. These remarks may but need not remain true for a brand new implementation of this utility; hence, these remarks have been moved into the RATIONALE.

BSD implementations historically required applications to provide the **–s** option whenever the archive was supposed to contain a symbol table. As in this volume of IEEE Std 1003.1-200x, System V historically creates or updates an archive symbol table whenever an object file is removed from, added to, or updated in the archive.

The OPERANDS section requires what might seem to be true without specifying it: the archive cannot truncate the filenames below {NAME_MAX}. Some historical implementations do so, however, causing unexpected results for the application. Therefore, this volume of IEEE Std 1003.1-200x makes the requirement explicit to avoid misunderstandings.

According to the System V documentation, the options **–dmpqrtx** are not required to begin with a hyphen (`'–'`). This volume of IEEE Std 1003.1-200x requires that a conforming application use the leading hyphen.

The archive format used by the 4.4 BSD implementation is documented in this RATIONALE as an example:

> A file created by *ar* begins with the ''magic'' string `"!<arch>\n"`. The rest of the archive is made up of objects, each of which is composed of a header for a file, a possible filename, and the file contents. The header is portable between machine architectures, and, if the file contents are printable, the archive is itself printable.
>
> The header is made up of six ASCII fields, followed by a two-character trailer. The fields are the object name (16 characters), the file last modification time (12 characters), the user and group IDs (each 6 characters), the file mode (8 characters), and the file size (10 characters). All numeric fields are in decimal, except for the file mode, which is in octal.
>
> The modification time is the file *st_mtime* field. The user and group IDs are the file *st_uid* and *st_gid* fields. The file mode is the file *st_mode* field. The file size is the file *st_size* field. The two-byte trailer is the string `"<newline>"`.
>
> Only the name field has any provision for overflow. If any filename is more than 16 characters in length or contains an embedded space, the string `"#1/"` followed by the ASCII length of the name is written in the name field. The file size (stored in the archive header) is incremented by the length of the name. The name is then written immediately following the archive header.
>
> Any unused characters in any of these fields are written as <space>s. If any fields are their particular maximum number of characters in length, there is no separation between the fields.
>
> Objects in the archive are always an even number of bytes long; files that are an odd number of bytes long are padded with a <newline>, although the size in the header does not reflect this.

The *ar* utility description requires that (when all its members are valid object files) *ar* produce an object code library, which the linkage editor can use to extract object modules. If the linkage editor needs a symbol table to permit random access to the archive, *ar* must provide it; however, *ar* does not require a symbol table.

The BSD **–o** option was omitted. It is a rare conforming application that uses *ar* to extract object code from a library with concern for its modification time, since this can only be of importance to *make*. Hence, since this functionality is not deemed important for applications portability, the modification time of the extracted files is set to the current time.

There is at least one known implementation (for a small computer) that can accommodate only object files for that system, disallowing mixed object and other files. The ability to handle any type of file is not only historical practice for most implementations, but is also a reasonable expectation.

Consideration was given to changing the output format of *ar* **–tv** to the same format as the output of *ls* **–l**. This would have made parsing the output of *ar* the same as that of *ls*. This was rejected in part because the current *ar* format is commonly used and changes would break historical usage. Second, *ar* gives the user ID and group ID in numeric format separated by a slash. Changing this to be the user name and group name would not be correct if the archive were moved to a machine that contained a different user database. Since *ar* cannot know whether the archive was generated on the same machine, it cannot tell what to report.

The text on the –**ur** option combination is historical practice—since one filename can easily represent two different files (for example, **/a/foo** and **/b/foo**), it is reasonable to replace the file in the archive even when the modification time in the archive is identical to that in the file system.

FUTURE DIRECTIONS

None.

SEE ALSO

c99, *pax*, *strip* the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 13, Headers, **<unistd.h>** description of {POSIX_NO_TRUNC}

CHANGE HISTORY

First released in Issue 2.

Issue 5

FUTURE DIRECTIONS section added.

Issue 6

This utility is now marked as part of the Software Development Utilities option.

The STDOUT description is changed for the –**v** option to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

IEEE PASC Interpretation 1003.2 #198 is applied, changing the description to consistently use ‘‘file’’ to refer to a file in the file system hierarchy, ‘‘archive’’ to refer to the archive being operated upon by the *ar* utility, and ‘‘file in the archive’’ to refer to a copy of a file that is contained in the archive.

NAME

asa — interpret carriage-control characters

SYNOPSIS

FR
```
asa [ file ... ]
```

DESCRIPTION

The *asa* utility shall write its input files to standard output, mapping carriage-control characters from the text files to line-printer control sequences in an implementation-defined manner.

The first character of every line shall be removed from the input, and the following actions are performed:

If the character removed is:

<space> The rest of the line is output without change.

0 A <newline> is output, then the rest of the input line.

1 One or more implementation-defined characters that causes an advance to the next page shall be output, followed by the rest of the input line.

+ The <newline> of the previous line shall be replaced with one or more implementation-defined characters that causes printing to return to column position 1, followed by the rest of the input line. If the '+' is the first character in the input, it shall be equivalent to <space>.

The action of the *asa* utility is unspecified upon encountering any character other than those listed above as the first character in a line.

OPTIONS

None.

OPERANDS

file A pathname of a text file used for input. If no *file* operands are specified, the standard input shall be used.

STDIN

The standard input shall be used only if no *file* operands are specified; see the INPUT FILES section.

INPUT FILES

The input files shall be text files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *asa*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
The standard output shall be the text from the input file modified as described in the DESCRIPTION section.

STDERR
None.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 All input files were output successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
None.

EXAMPLES

1. The following command:

   ```
   asa file
   ```

   permits the viewing of *file* (created by a program using FORTRAN-style carriage control characters) on a terminal.

2. The following command:

   ```
   a.out | asa | lp
   ```

   formats the FORTRAN output of **a.out** and directs it to the printer.

RATIONALE
The *asa* utility is needed to map ‘‘standard’’ FORTRAN 77 output into a form acceptable to contemporary printers. Usually, *asa* is used to pipe data to the *lp* utility; see *lp*.

This utility is generally used only by FORTRAN programs. The standard developers decided to retain *asa* to avoid breaking the historical large base of FORTRAN applications that put carriage-control characters in their output files. There is no requirement that a system have a FORTRAN compiler in order to run applications that need *asa*.

Historical implementations have used an ASCII <form-feed> in response to a 1 and an ASCII <carriage-return> in response to a `'+'`. It is suggested that implementations treat characters other than 0, 1, and `'+'` as <space> in the absence of any compelling reason to do otherwise. However, the action is listed here as ‘‘unspecified’’, permitting an implementation to provide

extensions to access fast multiple-line slewing and channel seeking in a non-portable manner.

FUTURE DIRECTIONS

None.

SEE ALSO

fort77, *lp*

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is now marked as part of the FORTRAN Runtime Utilities option.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

NAME

at — execute commands at a later time

SYNOPSIS

UP

```
at [–m][–f file][–q queuename] –t time_arg
at [–m][–f file][–q queuename] timespec ...
at –r at_job_id ...
at –l –q queuename
at –l [at_job_id ...]
```

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.

XSI

Users shall be permitted to use *at* if their name appears in the file **/usr/lib/cron/at.allow**. If that file does not exist, the file **/usr/lib/cron/at.deny** shall be checked to determine whether the user shall be denied access to *at*. If neither file exists, only a process with the appropriate privileges shall be allowed to submit a job. If only **at.deny** exists and is empty, global usage shall be permitted. The **at.allow** and **at.deny** files shall consist of one user name per line.

OPTIONS

The *at* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–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 shall be provided to the user by means of mail, unless they are redirected elsewhere; if there is no such output to provide, the implementation need not notify the user of the job's completion.

–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 *batch*. The meanings of all other *queuename*s are implementation-defined. If **–q** is specified along with either of the **–t** *time_arg* or *timespec* arguments, the results are unspecified.

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

–t *time_arg*
Submit the job to be run at the time specified by the *time* option-argument, which the application shall ensure has the format as specified by the *touch* **–t** *time* utility.

OPERANDS

The following operands shall be supported:

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

timespec
Submit the job to be run at the date and time specified. All of the *timespec* operands are interpreted as if they were separated by <space>s and concatenated, and shall be parsed as described in the grammar at the end of this section. The date and time shall be interpreted as being in the timezone of the user (as determined by the *TZ* variable), unless a timezone name appears as part of *time*, below.

In the POSIX locale, the following describes the three parts of the time 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-digit and two-digit numbers shall be taken to be hours; four-digit numbers to be hours and minutes. The time can alternatively 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 timezone name can also follow to further qualify the time. The acceptable timezone names are implementation-defined, except that they shall be case-insensitive and the string **utc** is supported to indicate the time is in Coordinated Universal Time. In the POSIX locale, the *time* field can also be one of the following tokens:

midnight
Indicates the time 12:00 am (00:00).

noon
Indicates the time 12:00 pm.

now
Indicates the current day and time. Invoking *at* <**now**> shall submit an at-job for potentially immediate execution (that is, 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). In the POSIX locale, two special days shall be recognized:

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
```

The following grammar describes the precise format of *timespec* in the POSIX locale. The general conventions for this style of grammar are described in Section 1.10 (on page 2220). This formal syntax shall take precedence over the preceding text syntax description. The longest possible token or delimiter shall be recognized at a given point. When used in a *timespec*, white space shall also delimit tokens.

```
%token hr24clock_hr_min
%token hr24clock_hour
/*
  A hr24clock_hr_min is a one, two, or four-digit number. A one-digit
  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 shall 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-digit, or four-digit number.
  A one-digit 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 shall 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.
```

```
5531          If no year_number is given, the current year is assumed if the given
5532          date and time are later this year. If no year_number is given and
5533          the date and time have already occurred this year and the month is
5534          not the current month, next year is the assumed year.
5535        */
5536        %token year_number
5537        /*
5538          A year_number is a four-digit number representing the year A.D., in
5539          which the at_job is to be run.
5540        */
5541        %token inc_number
5542        /*
5543          The inc_number is the number of times the succeeding increment
5544          period is to be added to the specified date and time.
5545        */
5546        %token timezone_name
5547        /*
5548          The name of an optional timezone suffix to the time field, in an
5549          implementation-defined format.
5550        */
5551        %token month_name
5552        /*
5553          One of the values from the mon or abmon keywords in the LC_TIME
5554          locale category.
5555        */
5556        %token day_of_week
5557        /*
5558          One of the values from the day or abday keywords in the LC_TIME
5559          locale category.
5560        */
5561        %token am_pm
5562        /*
5563          One of the values from the am_pm keyword in the LC_TIME locale
5564          category.
5565        */
5566        %start timespec
5567        %%
5568        timespec    : time
5569                    | time date
5570                    | time increment
5571                    | time date increment
5572                    | nowspec
5573                    ;
5574        nowspec     : "now"
5575                    | "now" increment
5576                    ;
5577        time        : hr24clock_hr_min
5578                    | 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"
                    ;

        date        : month_name day_number
                    | month_name day_number "," year_number
                    | day_of_week
                    | "today"
                    | "tomorrow"
                    ;

        increment   : "+" inc_number inc_period
                    | "next" inc_period
                    ;

        inc_period  : "minute" | "minutes"
                    | "hour" | "hours"
                    | "day" | "days"
                    | "week" | "weeks"
                    | "month" | "months"
                    | "year" | "years"
                    ;
```

STDIN

The standard input shall be a text file consisting of commands acceptable to the shell command language described in Chapter 2 (on page 2231). The standard input shall only be used if no **–f** *file* option is specified.

INPUT FILES

See the STDIN section.

XSI The text files **/usr/lib/cron/at.allow** and **/usr/lib/cron/at.deny** shall contain zero or more user names, one per line, of users who are, respectively, authorized or denied access to the *at* and *batch* utilities.

ENVIRONMENT VARIABLES

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

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of

diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

LC_TIME Determine the format and contents for date and time strings written and accepted by *at*.

SHELL Determine a name of a command interpreter to be used to invoke the at-job. If the variable is unset or null, *sh* shall be used. If it is set to a value other than a name for *sh*, the implementation shall do one of the following: use that shell; use *sh*; use the login shell from the user database; or any of the preceding accompanied by a warning diagnostic about which was chosen.

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

ASYNCHRONOUS EVENTS

Default.

STDOUT

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

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"
```

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

STDERR

In the POSIX locale, 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* has the same format as is described in the STDOUT section. 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.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 The *at* utility successfully submitted, removed, or listed a job or jobs.

>0 An error occurred.

CONSEQUENCES OF ERRORS

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

APPLICATION USAGE

The format of the *at* command line shown here is guaranteed only for the POSIX locale. Other cultures may be supported with substantially different interfaces, although implementations are encouraged to provide comparable levels of functionality.

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*. System V systems, for example, have used the login shell value for the user in **/etc/passwd**. To select reliably another command interpreter, the user must include it as part of the script, such as:

```
$ at 1800
myshell myscript
job ... at ...
$
```

EXAMPLES

1. This sequence can be used at a terminal:

   ```
   at –m 0730 tomorrow
   sort < file >outfile
   EOT
   ```

2. 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
   !
   ```

3. To have a job reschedule itself, *at* can be invoked from within the at-job. For example, this daily processing script named **my.daily** runs every day (although *crontab* is a more appropriate vehicle for such work):

   ```
   # my.daily runs every day
   daily processing
   at now tomorrow < my.daily
   ```

4. The spacing of the three portions of the POSIX locale *timespec* is quite flexible as long as there are no ambiguities. Examples of various times and operand presentation include:

   ```
   at 0815am Jan 24
   at 8 :15amjan24
   at now "+ 1day"
   at 5 pm FRIday
   at ’17
       utc+
       30minutes’
   ```

RATIONALE

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 specified commands.

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 proposals 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. The *at* functionality is useful in simple environments, but in large or complex situations, the functionality provided by the Batch Services option is more suitable.

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.

The behavior of *at* <*now*> was changed in an early proposal 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 backwards 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 that could not be easily parsed by shell scripts.

The *at* utility varies between System V and BSD systems in the way timezones 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 timezone default to that of the system, they merely need to issue the *at* command immediately following an unsetting or null assignment to *TZ*. For example:

```
TZ= at noon ...
```

gives the desired BSD result.

While the *yacc*-like grammar specified in the OPERANDS section 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*, and so on, based on the context.

FUTURE DIRECTIONS

None.

SEE ALSO

batch, *crontab*

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- If **–m** is not used, the job's standard output and standard error are provided to the user by mail.

The effects of using the **–q** and **–t** options as defined in the IEEE P1003.2b draft standard are specified.

The normative text is reworded to avoid use of the term ''must'' for application requirements.

NAME

awk — pattern scanning and processing language

SYNOPSIS

```
awk [–F ERE][–v assignment] ... program [argument ...]
awk [–F ERE] –f progfile ... [–v assignment] ...[argument ...]
```

DESCRIPTION

The *awk* utility shall execute programs written in the *awk* programming language, which is specialized for textual data manipulation. An *awk* program is a sequence of patterns and corresponding actions. When input is read that matches a pattern, the action associated with that pattern is carried out.

Input shall be interpreted as a sequence of records. By default, a record is a line, less its terminating <newline>, but this can be changed by using the **RS** built-in variable. Each record of input shall be matched in turn against each pattern in the program. For each pattern matched, the associated action shall be executed.

The *awk* utility shall interpret each input record as a sequence of fields where, by default, a field is a string of non-<blank>s. This default white-space field delimiter can be changed by using the **FS** built-in variable or the –**F** *ERE*. The *awk* utility shall denote the first field in a record $1, the second $2, and so on. The symbol $0 shall refer to the entire record; setting any other field causes the re-evaluation of $0. Assigning to $0 shall reset the values of all other fields and the **NF** built-in variable.

OPTIONS

The *awk* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–F *ERE* Define the input field separator to be the extended regular expression *ERE*, before any input is read; see **Regular Expressions** (on page 2363).

–f *progfile* Specify the pathname of the file *progfile* containing an *awk* program. If multiple instances of this option are specified, the concatenation of the files specified as *progfile* in the order specified shall be the *awk* program. The *awk* program can alternatively be specified in the command line as a single argument.

–v *assignment*
The application shall ensure that the *assignment* argument is in the same form as an *assignment* operand. The specified variable assignment shall occur prior to executing the *awk* program, including the actions associated with **BEGIN** patterns (if any). Multiple occurrences of this option can be specified.

OPERANDS

The following operands shall be supported:

program If no **–f** option is specified, the first operand to *awk* shall be the text of the *awk* program. The application shall supply the *program* operand as a single argument to *awk*. If the text does not end in a <newline>, *awk* shall interpret the text as if it did.

argument Either of the following two types of *argument* can be intermixed:

file A pathname of a file that contains the input to be read, which is matched against the set of patterns in the program. If no *file* operands are specified, or if a *file* operand is `'–'`, the standard input shall be used.

assignment An operand that begins with an underscore or alphabetic character from the portable character set (see the table in the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set), followed by a sequence of underscores, digits, and alphabetics from the portable character set, followed by the '=' character, shall specify a variable assignment rather than a pathname. The characters before the '=' represent the name of an *awk* variable; if that name is an *awk* reserved word (see **Grammar** (on page 2372)) the behavior is undefined. The characters following the equal sign shall be interpreted as if they appeared in the *awk* program preceded and followed by a double-quote ('"') character, as a **STRING** token (see **Grammar** (on page 2372)), except that if the last character is an unescaped backslash, it shall be interpreted as a literal backslash rather than as the first character of the sequence "\"". The variable shall be assigned the value of that **STRING** token and, if appropriate, shall be considered a *numeric string* (see **Expressions in awk** (on page 2358)), the variable shall also be assigned its numeric value. Each such variable assignment shall occur just prior to the processing of the following *file*, if any. Thus, an assignment before the first *file* argument shall be executed after the **BEGIN** actions (if any), while an assignment after the last *file* argument shall occur before the **END** actions (if any). If there are no *file* arguments, assignments shall be executed before processing the standard input.

STDIN

The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '–'; see the INPUT FILES section. If the *awk* program contains no actions and no patterns, but is otherwise a valid *awk* program, standard input and any *file* operands shall not be read and *awk* shall exit with a return status of zero.

INPUT FILES

Input files to the *awk* program from any of the following sources shall be text files:

- Any *file* operands or their equivalents, achieved by modifying the *awk* variables **ARGV** and **ARGC**
- Standard input in the absence of any *file* operands
- Arguments to the **getline** function

Whether the variable **RS** is set to a value other than a <newline> or not, for these files, implementations shall support records terminated with the specified separator up to {LINE_MAX} bytes and may support longer records.

If **–f** *progfile* is specified, the application shall ensure that the files named by each of the *progfile* option-arguments are text files containing an *awk* program.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *awk*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions and in comparisons of string values.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes within regular expressions, the identification of characters as letters, and the mapping of uppercase and lowercase characters for the **toupper** and **tolower** functions.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

LC_NUMERIC
Determine the radix character used when interpreting numeric input, performing conversions between numeric and string values, and formatting numeric output. Regardless of locale, the period character (the decimal-point character of the POSIX locale) is the decimal-point character recognized in processing *awk* programs (including assignments in command line arguments).

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

PATH Determine the search path when looking for commands executed by *system*(*expr*), or input and output pipes; see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

In addition, all environment variables shall be visible via the *awk* variable **ENVIRON**.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The nature of the output files depends on the *awk* program.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

The nature of the output files depends on the *awk* program.

EXTENDED DESCRIPTION

Overall Program Structure

An *awk* program is composed of pairs of the form:

```
pattern { action }
```

Either the pattern or the action (including the enclosing brace characters) can be omitted.

A missing pattern shall match any record of input, and a missing action shall be equivalent to:

```
{ print }
```

Execution of the *awk* program shall start by first executing the actions associated with all **BEGIN** patterns in the order they occur in the program. Then each *file* operand (or standard input if no files were specified) shall be processed in turn by reading data from the file until a record separator is seen (<newline> by default). Before the first reference to a field in the record is evaluated, the record shall be split into fields, according to the rules in **Regular Expressions** (on

page 2363), using the value of **FS** that was current at the time the record was read. Each pattern in the program then shall be evaluated in the order of occurrence, and the action associated with each pattern that matches the current record executed. The action for a matching pattern shall be executed before evaluating subsequent patterns. Finally, the actions associated with all **END** patterns shall be executed in the order they occur in the program.

Expressions in awk

Expressions describe computations used in *patterns* and *actions*. In the following table, valid expression operations are given in groups from highest precedence first to lowest precedence last, with equal-precedence operators grouped between horizontal lines. In expression evaluation, where the grammar is formally ambiguous, higher precedence operators shall be evaluated before lower precedence operators. In this table *expr*, *expr1*, *expr2*, and *expr3* represent any expression, while lvalue represents any entity that can be assigned to (that is, on the left side of an assignment operator). The precise syntax of expressions is given in **Grammar** (on page 2372).

Table 4-1 Expressions in Decreasing Precedence in *awk*

Syntax	Name	Type of Result	Associativity
(*expr*)	Grouping	Type of *expr*	N/A
$*expr*	Field reference	String	N/A
++ lvalue	Pre-increment	Numeric	N/A
-- lvalue	Pre-decrement	Numeric	N/A
lvalue ++	Post-increment	Numeric	N/A
lvalue --	Post-decrement	Numeric	N/A
expr ^ *expr*	Exponentiation	Numeric	Right
! *expr*	Logical not	Numeric	N/A
+ *expr*	Unary plus	Numeric	N/A
- *expr*	Unary minus	Numeric	N/A
expr * *expr*	Multiplication	Numeric	Left
expr / *expr*	Division	Numeric	Left
expr % *expr*	Modulus	Numeric	Left
expr + *expr*	Addition	Numeric	Left
expr - *expr*	Subtraction	Numeric	Left
expr *expr*	String concatenation	String	Left
expr < *expr*	Less than	Numeric	None
expr <= *expr*	Less than or equal to	Numeric	None
expr != *expr*	Not equal to	Numeric	None
expr == *expr*	Equal to	Numeric	None
expr > *expr*	Greater than	Numeric	None
expr >= *expr*	Greater than or equal to	Numeric	None
expr ~ *expr*	ERE match	Numeric	None
expr !~ *expr*	ERE non-match	Numeric	None

Syntax	Name	Type of Result	Associativity
expr in array	Array membership	Numeric	Left
(*index*) in *array*	Multi-dimension array membership	Numeric	Left
expr && *expr*	Logical AND	Numeric	Left
expr || *expr*	Logical OR	Numeric	Left
expr1 ? *expr2* : *expr3*	Conditional expression	Type of selected *expr2* or *expr3*	Right
lvalue ^= *expr*	Exponentiation assignment	Numeric	Right
lvalue %= *expr*	Modulus assignment	Numeric	Right
lvalue *= *expr*	Multiplication assignment	Numeric	Right
lvalue /= *expr*	Division assignment	Numeric	Right
lvalue += *expr*	Addition assignment	Numeric	Right
lvalue –= *expr*	Subtraction assignment	Numeric	Right
lvalue = *expr*	Assignment	Type of *expr*	Right

Each expression shall have either a string value, a numeric value, or both. Except as stated for specific contexts, the value of an expression shall be implicitly converted to the type needed for the context in which it is used. A string value shall be converted to a numeric value by the equivalent of the following calls to functions defined by the ISO C standard:

```
setlocale(LC_NUMERIC, "");
numeric_value = atof(string_value);
```

A numeric value that is exactly equal to the value of an integer (see Section 1.7.2 (on page 2207)) shall be converted to a string by the equivalent of a call to the **sprintf** function (see **String Functions** (on page 2369)) with the string "%d" as the *fmt* argument and the numeric value being converted as the first and only *expr* argument. Any other numeric value shall be converted to a string by the equivalent of a call to the **sprintf** function with the value of the variable **CONVFMT** as the *fmt* argument and the numeric value being converted as the first and only *expr* argument. The result of the conversion is unspecified if the value of **CONVFMT** is not a floating-point format specification. This volume of IEEE Std 1003.1-200x specifies no explicit conversions between numbers and strings. An application can force an expression to be treated as a number by adding zero to it, or can force it to be treated as a string by concatenating the null string ("") to it.

A string value shall be considered a *numeric string* if it comes from one of the following:

1. Field variables
2. Input from the *getline*( ) function
3. **FILENAME**
4. **ARGV** array elements
5. **ENVIRON** array elements
6. Array elements created by the *split*( ) function
7. A command line variable assignment
8. Variable assignment from another numeric string variable

and after all the following conversions have been applied, the resulting string would lexically be recognized as a **NUMBER** token as described by the lexical conventions in **Grammar** (on page

2372):

- All leading and trailing <blank>s are discarded
- If the first non-<blank> is `'+'` or `'-'`, it is discarded
- Changing each occurrence of the decimal point character from the current locale to a period

If a `'-'` character is ignored in the preceding description, the numeric value of the *numeric string* shall be the negation of the numeric value of the recognized **NUMBER** token. Otherwise, the numeric value of the *numeric string* shall be the numeric value of the recognized **NUMBER** token. Whether or not a string is a *numeric string* shall be relevant only in contexts where that term is used in this section.

When an expression is used in a Boolean context, if it has a numeric value, a value of zero shall be treated as false and any other value shall be treated as true. Otherwise, a string value of the null string shall be treated as false and any other value shall be treated as true. A Boolean context shall be one of the following:

- The first subexpression of a conditional expression
- An expression operated on by logical NOT, logical AND, or logical OR
- The second expression of a **for** statement
- The expression of an **if** statement
- The expression of the **while** clause in either a **while** or **do**...**while** statement
- An expression used as a pattern (as in Overall Program Structure)

All arithmetic shall follow the semantics of floating-point arithmetic as specified by the ISO C standard (see Section 1.7.2 (on page 2207)).

The value of the expression:

expr1 `^` *expr2*

shall be equivalent to the value returned by the ISO C standard function call:

`pow(`*expr1*`, `*expr2*`)`

The expression:

lvalue `^=` *expr*

shall be equivalent to the ISO C standard expression:

lvalue `= pow(`*lvalue*`, `*expr*`)`

except that lvalue shall be evaluated only once. The value of the expression:

expr1 `%` *expr2*

shall be equivalent to the value returned by the ISO C standard function call:

`fmod(`*expr1*`, `*expr2*`)`

The expression:

lvalue `%=` *expr*

shall be equivalent to the ISO C standard expression:

lvalue `= fmod(`*lvalue*`, `*expr*`)`

except that lvalue shall be evaluated only once.

Variables and fields shall be set by the assignment statement:

```
lvalue = expression
```

and the type of *expression* shall determine the resulting variable type. The assignment includes the arithmetic assignments ("+=", "–=", "*=", "/=", "%=", "^=", "++", "––") all of which shall produce a numeric result. The left-hand side of an assignment and the target of increment and decrement operators can be one of a variable, an array with index, or a field selector.

The *awk* language supplies arrays that are used for storing numbers or strings. Arrays need not be declared. They shall initially be empty, and their sizes shall change dynamically. The subscripts, or element identifiers, are strings, providing a type of associative array capability. An array name followed by a subscript within square brackets can be used as an lvalue and thus as an expression, as described in the grammar; see **Grammar** (on page 2372). Unsubscripted array names can be used in only the following contexts:

- A parameter in a function definition or function call
- The **NAME** token following any use of the keyword **in** as specified in the grammar (see **Grammar** (on page 2372)); if the name used in this context is not an array name, the behavior is undefined

A valid array *index* shall consist of one or more comma-separated expressions, similar to the way in which multi-dimensional arrays are indexed in some programming languages. Because *awk* arrays are really one-dimensional, such a comma-separated list shall be converted to a single string by concatenating the string values of the separate expressions, each separated from the other by the value of the **SUBSEP** variable. Thus, the following two index operations shall be equivalent:

```
var[expr1, expr2, ... exprn]
var[expr1 SUBSEP expr2 SUBSEP ... SUBSEP exprn]
```

The application shall ensure that a multi-dimensioned *index* used with the **in** operator is parenthesized. The **in** operator, which tests for the existence of a particular array element, shall not cause that element to exist. Any other reference to a nonexistent array element shall automatically create it.

Comparisons (with the '<', "<=", "!=", "==", '>', and ">=" operators) shall be made numerically if both operands are numeric, if one is numeric and the other has a string value that is a numeric string, or if one is numeric and the other has the uninitialized value. Otherwise, operands shall be converted to strings as required and a string comparison shall be made using the locale-specific collation sequence. The value of the comparison expression shall be 1 if the relation is true, or 0 if the relation is false.

Variables and Special Variables

Variables can be used in an *awk* program by referencing them. With the exception of function parameters (see **User-Defined Functions** (on page 2371)), they are not explicitly declared. Function parameter names shall be local to the function; all other variable names shall be global. The same name shall not be used as both a function parameter name and as the name of a function or a special *awk* variable. The same name shall not be used both as a variable name with global scope and as the name of a function. The same name shall not be used within the same scope both as a scalar variable and as an array. Uninitialized variables, including scalar variables, array elements, and field variables, shall have an uninitialized value. An uninitialized value shall have both a numeric value of zero and a string value of the empty string. Evaluation

of variables with an uninitialized value, to either string or numeric, shall be determined by the context in which they are used.

Field variables shall be designated by a '$' followed by a number or numerical expression. The effect of the field number *expression* evaluating to anything other than a non-negative integer is unspecified; uninitialized variables or string values need not be converted to numeric values in this context. New field variables can be created by assigning a value to them. References to nonexistent fields (that is, fields after $**NF**), shall evaluate to the uninitialized value. Such references shall not create new fields. However, assigning to a nonexistent field (for example, $(**NF**+2)=5) shall increase the value of **NF**; create any intervening fields with the uninitialized value; and cause the value of $0 to be recomputed, with the fields being separated by the value of **OFS**. Each field variable shall have a string value or an uninitialized value when created. Field variables shall have the uninitialized value when created from $0 using **FS** and the variable does not contain any characters. If appropriate, the field variable shall be considered a numeric string (see **Expressions in awk** (on page 2358)).

Implementations shall support the following other special variables that are set by *awk*:

ARGC The number of elements in the **ARGV** array.

ARGV An array of command line arguments, excluding options and the *program* argument, numbered from zero to **ARGC**–1.

The arguments in **ARGV** can be modified or added to; **ARGC** can be altered. As each input file ends, *awk* shall treat the next non-null element of **ARGV**, up to the current value of **ARGC**–1, inclusive, as the name of the next input file. Thus, setting an element of **ARGV** to null means that it shall not be treated as an input file. The name '–' indicates the standard input. If an argument matches the format of an *assignment* operand, this argument shall be treated as an assignment rather than a *file* argument.

CONVFMT The **printf** format for converting numbers to strings (except for output statements, where **OFMT** is used); "%.6g" by default.

ENVIRON An array representing the value of the environment, as described in the *exec* functions defined in the System Interfaces volume of IEEE Std 1003.1-200x. The indices of the array shall be strings consisting of the names of the environment variables, and the value of each array element shall be a string consisting of the value of that variable. If appropriate, the environment variable shall be considered a *numeric string* (see **Expressions in awk** (on page 2358)), the array element shall also have its numeric value.

In all cases where the behavior of *awk* is affected by environment variables (including the environment of any commands that *awk* executes via the **system** function or via pipeline redirections with the **print** statement, the **printf** statement, or the **getline** function), the environment used shall be the environment at the time *awk* began executing; it is implementation-defined whether any modification of **ENVIRON** affects this environment.

FILENAME A pathname of the current input file. Inside a **BEGIN** action the value is undefined. Inside an **END** action the value shall be the name of the last input file processed.

FNR The ordinal number of the current record in the current file. Inside a **BEGIN** action the value shall be zero. Inside an **END** action the value shall be the number of the last record processed in the last file processed.

FS Input field separator regular expression; a <space> by default.

NF The number of fields in the current record. Inside a **BEGIN** action, the use of **NF** is undefined unless a **getline** function without a *var* argument is executed previously. Inside an **END** action, **NF** shall retain the value it had for the last record read, unless a subsequent, redirected, **getline** function without a *var* argument is performed prior to entering the **END** action.

NR The ordinal number of the current record from the start of input. Inside a **BEGIN** action the value shall be zero. Inside an **END** action the value shall be the number of the last record processed.

OFMT The **printf** format for converting numbers to strings in output statements (see **Output Statements** (on page 2367)); `"%.6g"` by default. The result of the conversion is unspecified if the value of **OFMT** is not a floating-point format specification.

OFS The **print** statement output field separation; <space> by default.

ORS The **print** statement output record separator; a <newline> by default.

RLENGTH The length of the string matched by the **match** function.

RS The first character of the string value of **RS** shall be the input record separator; a <newline> by default. If **RS** contains more than one character, the results are unspecified. If **RS** is null, then records are separated by sequences consisting of a <newline> plus one or more blank lines, leading or trailing blank lines shall not result in empty records at the beginning or end of the input, and a <newline> shall always be a field separator, no matter what the value of **FS** is.

RSTART The starting position of the string matched by the **match** function, numbering from 1. This shall always be equivalent to the return value of the **match** function.

SUBSEP The subscript separator string for multi-dimensional arrays; the default value is implementation-defined.

Regular Expressions

The *awk* utility shall make use of the extended regular expression notation (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.4, Extended Regular Expressions) except that it shall allow the use of C-language conventions for escaping special characters within the EREs, as specified in the table in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`) and the following table; these escape sequences shall be recognized both inside and outside bracket expressions. Note that records need not be separated by <newline>s and string constants can contain <newline>s, so even the `"\n"` sequence is valid in *awk* EREs. Using a slash character within an ERE requires the escaping shown in the following table.

Table 4-2 Escape Sequences in *awk*

Escape Sequence	Description	Meaning
`\"`	Backslash quotation-mark	Quotation-mark character
`\/`	Backslash slash	Slash character
`\ddd`	A backslash character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the one, two, or three-digit octal integer. If the size of a byte on the system is greater than nine bits, the valid escape sequence used to represent a byte is implementation-defined. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading `'\'` for each byte.
`\c`	A backslash character followed by any character not described in this table or in the table in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`)	Undefined

A regular expression can be matched against a specific field or string by using one of the two regular expression matching operators, `'~'` and `"!~"`. These operators shall interpret their right-hand operand as a regular expression and their left-hand operand as a string. If the regular expression matches the string, the `'~'` expression shall evaluate to a value of 1, and the `"!~"` expression shall evaluate to a value of 0. (The regular expression matching operation is as defined by the term matched in the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.1, Regular Expression Definitions, where a match occurs on any part of the string unless the regular expression is limited with the circumflex or dollar sign special characters.) If the regular expression does not match the string, the `'~'` expression shall evaluate to a value of 0, and the `"!~"` expression shall evaluate to a value of 1. If the right-hand operand is any expression other than the lexical token **ERE**, the string value of the expression shall be interpreted as an extended regular expression, including the escape conventions described above. Note that these same escape conventions shall also be applied in determining the value of a string literal (the lexical token **STRING**), and thus shall be applied a second time when a string literal is used in this context.

When an **ERE** token appears as an expression in any context other than as the right-hand of the `'~'` or `"!~"` operator or as one of the built-in function arguments described below, the value of the resulting expression shall be the equivalent of:

```
$0 ~ /ere/
```

The *ere* argument to the **gsub**, **match**, **sub** functions, and the *fs* argument to the **split** function (see **String Functions** (on page 2369)) shall be interpreted as extended regular expressions. These can be either **ERE** tokens or arbitrary expressions, and shall be interpreted in the same manner as the right-hand side of the `'~'` or `"!~"` operator.

An extended regular expression can be used to separate fields by using the **–F** *ERE* option or by assigning a string containing the expression to the built-in variable **FS**. The default value of the **FS** variable shall be a single <space>. The following describes **FS** behavior:

1. If **FS** is a null string, the behavior is unspecified.
2. If **FS** is a single character:
   a. If **FS** is <space>, skip leading and trailing <blank>s; fields shall be delimited by sets of one or more <blank>s.
   b. Otherwise, if **FS** is any other character *c*, fields shall be delimited by each single occurrence of *c*.
3. Otherwise, the string value of **FS** shall be considered to be an extended regular expression. Each occurrence of a sequence matching the extended regular expression shall delimit fields.

Except for the '~' and "!~" operators, and in the **gsub**, **match**, **split**, and **sub** built-in functions, ERE matching shall be based on input records; that is, record separator characters (the first character of the value of the variable **RS**, <newline> by default) cannot be embedded in the expression, and no expression shall match the record separator character. If the record separator is not <newline>, <newline>s embedded in the expression can be matched. For the '~' and "!~" operators, and in those four built-in functions, ERE matching shall be based on text strings; that is, any character (including <newline> and the record separator) can be embedded in the pattern, and an appropriate pattern shall match any character. However, in all *awk* ERE matching, the use of one or more NUL characters in the pattern, input record, or text string produces undefined results.

Patterns

A *pattern* is any valid *expression*, a range specified by two expressions separated by comma, or one of the two special patterns **BEGIN** or **END**.

Special Patterns

The *awk* utility shall recognize two special patterns, **BEGIN** and **END**. Each **BEGIN** pattern shall be matched once and its associated action executed before the first record of input is read (except possibly by use of the **getline** function—see **Input/Output and General Functions** (on page 2370)—in a prior **BEGIN** action) and before command line assignment is done. Each **END** pattern shall be matched once and its associated action executed after the last record of input has been read. These two patterns shall have associated actions.

BEGIN and **END** shall not combine with other patterns. Multiple **BEGIN** and **END** patterns shall be allowed. The actions associated with the **BEGIN** patterns shall be executed in the order specified in the program, as are the **END** actions. An **END** pattern can precede a **BEGIN** pattern in a program.

If an *awk* program consists of only actions with the pattern **BEGIN**, and the **BEGIN** action contains no **getline** function, *awk* shall exit without reading its input when the last statement in the last **BEGIN** action is executed. If an *awk* program consists of only actions with the pattern **END** or only actions with the patterns **BEGIN** and **END**, the input shall be read before the statements in the **END** actions are executed.

Expression Patterns

An expression pattern shall be evaluated as if it were an expression in a Boolean context. If the result is true, the pattern shall be considered to match, and the associated action (if any) shall be executed. If the result is false, the action shall not be executed.

Pattern Ranges

A pattern range consists of two expressions separated by a comma; in this case, the action shall be performed for all records between a match of the first expression and the following match of the second expression, inclusive. At this point, the pattern range can be repeated starting at input records subsequent to the end of the matched range.

Actions

An action is a sequence of statements as shown in the grammar in **Grammar** (on page 2372). Any single statement can be replaced by a statement list enclosed in braces. The application shall ensure that statements in a statement list are separated by <newline>s or semicolons. Statements in a statement list shall be executed sequentially in the order that they appear.

The *expression* acting as the conditional in an **if** statement shall be evaluated and if it is non-zero or non-null, the following *statement* shall be executed; otherwise, if **else** is present, the statement following the **else** shall be executed.

The **if**, **while**, **do**...**while**, **for**, **break**, and **continue** statements are based on the ISO C standard (see Section 1.7.2 (on page 2207)), except that the Boolean expressions shall be treated as described in **Expressions in awk** (on page 2358), and except in the case of:

```
for (variable in array)
```

which shall iterate, assigning each *index* of *array* to *variable* in an unspecified order. The results of adding new elements to *array* within such a **for** loop are undefined. If a **break** or **continue** statement occurs outside of a loop, the behavior is undefined.

The **delete** statement shall remove an individual array element. Thus, the following code deletes an entire array:

```
for (index in array)
    delete array[index]
```

The **next** statement shall cause all further processing of the current input record to be abandoned. The behavior is undefined if a **next** statement appears or is invoked in a **BEGIN** or **END** action.

The **exit** statement shall invoke all **END** actions in the order in which they occur in the program source and then terminate the program without reading further input. An **exit** statement inside an **END** action shall terminate the program without further execution of **END** actions. If an expression is specified in an **exit** statement, its numeric value shall be the exit status of *awk*, unless subsequent errors are encountered or a subsequent **exit** statement with an expression is executed.

6274 **Output Statements**

6275 Both **print** and **printf** statements shall write to standard output by default. The output shall be
6276 written to the location specified by *output_redirection* if one is supplied, as follows:

6277 `> expression`
6278 `>> expression`
6279 `| expression`

6280 In all cases, the *expression* shall be evaluated to produce a string that is used as a pathname into
6281 which to write (for `'>'` or `">>"`) or as a command to be executed (for `'|'`). Using the first two
6282 forms, if the file of that name is not currently open, it shall be opened, creating it if necessary and
6283 using the first form, truncating the file. The output then shall be appended to the file. As long as
6284 the file remains open, subsequent calls in which *expression* evaluates to the same string value
6285 shall simply append output to the file. The file remains open until the **close** function (see
6286 **Input/Output and General Functions** (on page 2370)) is called with an expression that evaluates
6287 to the same string value.

6288 The third form shall write output onto a stream piped to the input of a command. The stream
6289 shall be created if no stream is currently open with the value of *expression* as its command name.
6290 The stream created shall be equivalent to one created by a call to the *popen*( ) function defined in
6291 the System Interfaces volume of IEEE Std 1003.1-200x with the value of *expression* as the
6292 *command* argument and a value of *w* as the *mode* argument. As long as the stream remains open,
6293 subsequent calls in which *expression* evaluates to the same string value shall write output to the
6294 existing stream. The stream shall remain open until the **close** function (see **Input/Output and**
6295 **General Functions** (on page 2370)) is called with an expression that evaluates to the same string
6296 value. At that time, the stream shall be closed as if by a call to the *pclose*( ) function defined in
6297 the System Interfaces volume of IEEE Std 1003.1-200x.

6298 As described in detail by the grammar in **Grammar** (on page 2372), these output statements shall
6299 take a comma-separated list of *expression*s referred to in the grammar by the non-terminal
6300 symbols **expr_list**, **print_expr_list**, or **print_expr_list_opt**. This list is referred to here as the
6301 *expression list*, and each member is referred to as an *expression argument*.

6302 The **print** statement shall write the value of each expression argument onto the indicated output
6303 stream separated by the current output field separator (see variable **OFS** above), and terminated
6304 by the output record separator (see variable **ORS** above). All expression arguments shall be
6305 taken as strings, being converted if necessary; this conversion shall be as described in
6306 **Expressions in awk** (on page 2358), with the exception that the **printf** format in **OFMT** shall be
6307 used instead of the value in **CONVFMT**. An empty expression list shall stand for the whole
6308 input record ($0).

6309 The **printf** statement shall produce output based on a notation similar to the File Format
6310 Notation used to describe file formats in this volume of IEEE Std 1003.1-200x (see the Base
6311 Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation). Output shall be
6312 produced as specified with the first expression argument as the string *format* and subsequent
6313 expression arguments as the strings *arg1* to *argn*, inclusive, with the following exceptions:

6314 1. The *format* shall be an actual character string rather than a graphical representation.
6315 Therefore, it cannot contain empty character positions. The <space> in the *format* string, in
6316 any context other than a *flag* of a conversion specification, shall be treated as an ordinary
6317 character that is copied to the output.

6318 2. If the character set contains a `'Δ'` character and that character appears in the *format* string,
6319 it shall be treated as an ordinary character that is copied to the output.

3. The *escape sequences* beginning with a backslash character shall be treated as sequences of ordinary characters that are copied to the output. Note that these same sequences shall be interpreted lexically by *awk* when they appear in literal strings, but they shall not be treated specially by the **printf** statement.

4. A *field width* or *precision* can be specified as the '*' character instead of a digit string. In this case the next argument from the expression list shall be fetched and its numeric value taken as the field width or precision.

5. The implementation shall not precede or follow output from the *d* or *u* conversion specifications with <blank>s not specified by the *format* string.

6. The implementation shall not precede output from the *o* conversion specification with leading zeros not specified by the *format* string.

7. For the *c* conversion specification: if the argument has a numeric value, the character whose encoding is that value shall be output. If the value is zero or is not the encoding of any character in the character set, the behavior is undefined. If the argument does not have a numeric value, the first character of the string value shall be output; if the string does not contain any characters, the behavior is undefined.

8. For each conversion specification that consumes an argument, the next expression argument shall be evaluated. With the exception of the *c* conversion, the value shall be converted (according to the rules specified in **Expressions in awk** (on page 2358)) to the appropriate type for the conversion specification.

9. If there are insufficient expression arguments to satisfy all the conversion specifications in the *format* string, the behavior is undefined.

10. If any character sequence in the *format* string begins with a '%' character, but does not form a valid conversion specification, the behavior is unspecified.

Both **print** and **printf** can output at least {LINE_MAX} bytes.

Functions

The *awk* language has a variety of built-in functions: arithmetic, string, input/output, and general.

Arithmetic Functions

The arithmetic functions, except for **int**, shall be based on the ISO C standard (see Section 1.7.2 (on page 2207)). The behavior is undefined in cases where the ISO C standard specifies that an error be returned or that the behavior is undefined. Although the grammar (see **Grammar** (on page 2372)) permits built-in functions to appear with no arguments or parentheses, unless the argument or parentheses are indicated as optional in the following list (by displaying them within the "[]" brackets), such use is undefined.

atan2(*y*,*x*) Return arctangent of *y*/*x* in radians in the range [–π,π].

cos(*x*) Return cosine of *x*, where *x* is in radians.

sin(*x*) Return sine of *x*, where *x* is in radians.

exp(*x*) Return the exponential function of *x*.

log(*x*) Return the natural logarithm of *x*.

sqrt(*x*) Return the square root of *x*.

int(*x*) Return the argument truncated to an integer. Truncation shall be toward 0 when *x*>0.

rand( ) Return a random number *n*, such that 0≤*n*<1.

srand([*expr***])** Set the seed value for *rand* to *expr* or use the time of day if *expr* is omitted. The previous seed value shall be returned.

String Functions

The string functions in the following list shall be supported. Although the grammar (see **Grammar** (on page 2372)) permits built-in functions to appear with no arguments or parentheses, unless the argument or parentheses are indicated as optional in the following list (by displaying them within the "[]" brackets), such use is undefined.

gsub(*ere*, *repl***[**, *in***]**)
Behave like **sub** (see below), except that it shall replace all occurrences of the regular expression (like the *ed* utility global substitute) in $0 or in the *in* argument, when specified.

index(*s*, *t*) Return the position, in characters, numbering from 1, in string *s* where string *t* first occurs, or zero if it does not occur at all.

length[([*s***])]** Return the length, in characters, of its argument taken as a string, or of the whole record, $0, if there is no argument.

match(*s*, *ere*) Return the position, in characters, numbering from 1, in string *s* where the extended regular expression *ere* occurs, or zero if it does not occur at all. RSTART shall be set to the starting position (which is the same as the returned value), zero if no match is found; RLENGTH shall be set to the length of the matched string, –1 if no match is found.

split(*s*, *a***[**, *fs* **]**)
Split the string *s* into array elements *a*[1], *a*[2], ..., *a*[*n*], and return *n*. All elements of the array shall be deleted before the split is performed. The separation shall be done with the ERE *fs* or with the field separator **FS** if *fs* is not given. Each array element shall have a string value when created and, if appropriate, the array element shall be considered a numeric string (see **Expressions in awk** (on page 2358)). The effect of a null string as the value of *fs* is unspecified.

sprintf(*fmt*, *expr*, *expr*, ...)
Format the expressions according to the **printf** format given by *fmt* and return the resulting string.

sub(*ere*, *repl***[**, *in* **])**
Substitute the string *repl* in place of the first instance of the extended regular expression *ERE* in string *in* and return the number of substitutions. An ampersand ('&') appearing in the string *repl* shall be replaced by the string from *in* that matches the ERE. An ampersand preceded with a backslash ('\') shall be interpreted as the literal ampersand character. An occurrence of two consecutive backslashes shall be interpreted as just a single literal backslash character. Any other occurrence of a backslash (for example, preceding any other character) shall be treated as a literal backslash character. Note that if *repl* is a string literal (the lexical token **STRING**; see **Grammar** (on page 2372)), the handling of the ampersand character occurs after any lexical processing, including any lexical backslash escape sequence processing. If *in* is specified and it is not an lvalue (see **Expressions in awk** (on page 2358)), the behavior is undefined. If *in* is omitted, *awk*

shall use the current record ($0) in its place.

substr(*s*, *m*[, *n*])
: Return the at most *n*-character substring of *s* that begins at position *m*, numbering from 1. If *n* is omitted, or if *n* specifies more characters than are left in the string, the length of the substring shall be limited by the length of the string *s*.

tolower(*s*)
: Return a string based on the string *s*. Each character in *s* that is an uppercase letter specified to have a **tolower** mapping by the *LC_CTYPE* category of the current locale shall be replaced in the returned string by the lowercase letter specified by the mapping. Other characters in *s* shall be unchanged in the returned string.

toupper(*s*)
: Return a string based on the string *s*. Each character in *s* that is a lowercase letter specified to have a **toupper** mapping by the *LC_CTYPE* category of the current locale is replaced in the returned string by the uppercase letter specified by the mapping. Other characters in *s* are unchanged in the returned string.

All of the preceding functions that take *ERE* as a parameter expect a pattern or a string valued expression that is a regular expression as defined in **Regular Expressions** (on page 2363).

Input/Output and General Functions

The input/output and general functions are:

close(*expression*)
: Close the file or pipe opened by a **print** or **printf** statement or a call to **getline** with the same string-valued *expression*. The limit on the number of open *expression* arguments is implementation-defined. If the close was successful, the function shall return zero; otherwise, it shall return non-zero.

expression | **getline** [*var*]
: Read a record of input from a stream piped from the output of a command. The stream shall be created if no stream is currently open with the value of *expression* as its command name. The stream created shall be equivalent to one created by a call to the *popen*( ) function with the value of *expression* as the *command* argument and a value of *r* as the *mode* argument. As long as the stream remains open, subsequent calls in which *expression* evaluates to the same string value shall read subsequent records from the stream. The stream shall remain open until the **close** function is called with an expression that evaluates to the same string value. At that time, the stream shall be closed as if by a call to the *pclose*( ) function. If *var* is omitted, $0 and **NF** shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered a numeric string (see **Expressions in awk** (on page 2358)).

  The **getline** operator can form ambiguous constructs when there are unparenthesized operators (including concatenate) to the left of the `'|'` (to the beginning of the expression containing **getline**). In the context of the `'$'` operator, `'|'` shall behave as if it had a lower precedence than `'$'`. The result of evaluating other operators is unspecified, and conforming applications shall parenthesize properly all such usages.

getline
: Set $0 to the next input record from the current input file. This form of **getline** shall set the **NF**, **NR**, and **FNR** variables.

getline *var*
: Set variable *var* to the next input record from the current input file and, if appropriate, *var* shall be considered a numeric string (see **Expressions in awk** (on page 2358)). This form of **getline** shall set the **FNR** and **NR** variables.

getline [*var***]** < *expression*

Read the next record of input from a named file. The *expression* shall be evaluated to produce a string that is used as a pathname. If the file of that name is not currently open, it shall be opened. As long as the stream remains open, subsequent calls in which *expression* evaluates to the same string value shall read subsequent records from the file. The file shall remain open until the **close** function is called with an expression that evaluates to the same string value. If *var* is omitted, $0 and **NF** shall be set; otherwise, *var* shall be set and, if appropriate, it shall be considered a numeric string (see **Expressions in awk** (on page 2358)).

The **getline** operator can form ambiguous constructs when there are unparenthesized binary operators (including concatenate) to the right of the '<' (up to the end of the expression containing the **getline**). The result of evaluating such a construct is unspecified, and conforming applications shall parenthesize properly all such usages.

system(*expression*)

Execute the command given by *expression* in a manner equivalent to the *system*() function defined in the System Interfaces volume of IEEE Std 1003.1-200x and return the exit status of the command.

All forms of **getline** shall return 1 for successful input, zero for end-of-file, and –1 for an error.

Where strings are used as the name of a file or pipeline, the application shall ensure that the strings are textually identical. The terminology ‘‘same string value’’ implies that ‘‘equivalent strings’’, even those that differ only by <space>s, represent different files.

User-Defined Functions

The *awk* language also provides user-defined functions. Such functions can be defined as:

```
function name([parameter, ...]) { statements }
```

A function can be referred to anywhere in an *awk* program; in particular, its use can precede its definition. The scope of a function is global.

Function parameters, if present, can be either scalars or arrays; the behavior is undefined if an array name is passed as a parameter that the function uses as a scalar, or if a scalar expression is passed as a parameter that the function uses as an array. Function parameters shall be passed by value if scalar and by reference if array name.

The number of parameters in the function definition need not match the number of parameters in the function call. Excess formal parameters can be used as local variables. If fewer arguments are supplied in a function call than are in the function definition, the extra parameters that are used in the function body as scalars shall evaluate to the uninitialized value until they are otherwise initialized, and the extra parameters that are used in the function body as arrays shall be treated as uninitialized arrays where each element evaluates to the uninitialized value until otherwise initialized.

When invoking a function, no white space can be placed between the function name and the opening parenthesis. Function calls can be nested and recursive calls can be made upon functions. Upon return from any nested or recursive function call, the values of all of the calling function’s parameters shall be unchanged, except for array parameters passed by reference. The **return** statement can be used to return a value. If a **return** statement appears outside of a function definition, the behavior is undefined.

In the function definition, <newline>s shall be optional before the opening brace and after the closing brace. Function definitions can appear anywhere in the program where a *pattern-action*

pair is allowed.

Grammar

The grammar in this section and the lexical conventions in the following section shall together describe the syntax for *awk* programs. The general conventions for this style of grammar are described in Section 1.10 (on page 2220). A valid program can be represented as the non-terminal symbol *program* in the grammar. This formal syntax shall take precedence over the preceding text syntax description.

```
%token NAME NUMBER STRING ERE
%token FUNC_NAME   /* Name followed by '(' without white space. */

/* Keywords  */
%token       Begin   End
/*          'BEGIN' 'END'                              */

%token       Break   Continue   Delete   Do   Else
/*          'break' 'continue' 'delete' 'do' 'else'  */

%token       Exit   For   Function   If   In
/*          'exit' 'for' 'function' 'if' 'in'        */

%token       Next   Print   Printf   Return   While
/*          'next' 'print' 'printf' 'return' 'while' */

/* Reserved function names */
%token BUILTIN_FUNC_NAME
            /* One token for the following:
             * atan2 cos sin exp log sqrt int rand srand
             * gsub index length match split sprintf sub
             * substr tolower toupper close system
             */
%token GETLINE
            /* Syntactically different from other built-ins. */

/* Two-character tokens. */
%token ADD_ASSIGN SUB_ASSIGN MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN POW_ASSIGN
/*     '+='       '-='       '*='       '/='       '%='       '^=' */

%token OR   AND  NO_MATCH   EQ   LE   GE   NE   INCR  DECR  APPEND
/*     '||' '&&' '!~' '==' '<=' '>=' '!=' '++'  '--'  '>>'   */

/* One-character tokens. */
%token '{' '}' '(' ')' '[' ']' ',' ';' NEWLINE
%token '+' '-' '*' '%' '^' '!' '>' '<' '|' '?' ':' '~' '$' '='

%start program
%%

program          : item_list
                 | actionless_item_list
                 ;

item_list        : newline_opt
                 | actionless_item_list item terminator
                 | item_list            item terminator
                 | item_list          action terminator
                 ;
```

```
6543        actionless_item_list : item_list            pattern terminator
6544                             | actionless_item_list pattern terminator
6545                             ;
6546        item                 : pattern action
6547                             | Function NAME      '(' param_list_opt ')'
6548                                 newline_opt action
6549                             | Function FUNC_NAME '(' param_list_opt ')'
6550                                 newline_opt action
6551                             ;
6552        param_list_opt       : /* empty */
6553                             | param_list
6554                             ;
6555        param_list           : NAME
6556                             | param_list ',' NAME
6557                             ;
6558        pattern              : Begin
6559                             | End
6560                             | expr
6561                             | expr ',' newline_opt expr
6562                             ;
6563        action               : '{' newline_opt                               '}'
6564                             | '{' newline_opt terminated_statement_list   '}'
6565                             | '{' newline_opt unterminated_statement_list '}'
6566                             ;
6567        terminator           : terminator ';'
6568                             | terminator NEWLINE
6569                             |            ';'
6570                             |            NEWLINE
6571                             ;
6572        terminated_statement_list : terminated_statement
6573                             | terminated_statement_list terminated_statement
6574                             ;
6575        unterminated_statement_list : unterminated_statement
6576                             | terminated_statement_list unterminated_statement
6577                             ;
6578        terminated_statement : action newline_opt
6579                             | If '(' expr ')' newline_opt terminated_statement
6580                             | If '(' expr ')' newline_opt terminated_statement
6581                                  Else newline_opt terminated_statement
6582                             | While '(' expr ')' newline_opt terminated_statement
6583                             | For '(' simple_statement_opt ';'
6584                                 expr_opt ';' simple_statement_opt ')' newline_opt
6585                                 terminated_statement
6586                             | For '(' NAME In NAME ')' newline_opt
6587                                 terminated_statement
6588                             | ';' newline_opt
6589                             | terminatable_statement NEWLINE newline_opt
6590                             | terminatable_statement ';'     newline_opt
```

```
                    ;
unterminated_statement : terminatable_statement
                    | If '(' expr ')' newline_opt unterminated_statement
                    | If '(' expr ')' newline_opt terminated_statement
                        Else newline_opt unterminated_statement
                    | While '(' expr ')' newline_opt unterminated_statement
                    | For '(' simple_statement_opt ';'
                     expr_opt ';' simple_statement_opt ')' newline_opt
                        unterminated_statement
                    | For '(' NAME In NAME ')' newline_opt
                        unterminated_statement
                    ;
terminatable_statement : simple_statement
                    | Break
                    | Continue
                    | Next
                    | Exit expr_opt
                    | Return expr_opt
                    | Do newline_opt terminated_statement While '(' expr ')'
                    ;
simple_statement_opt : /* empty */
                    | simple_statement
                    ;
simple_statement : Delete NAME '[' expr_list ']'
                    | expr
                    | print_statement
                    ;
print_statement  : simple_print_statement
                    | simple_print_statement output_redirection
                    ;
simple_print_statement : Print  print_expr_list_opt
                    | Print  '(' multiple_expr_list ')'
                    | Printf print_expr_list
                    | Printf '(' multiple_expr_list ')'
                    ;
output_redirection : '>'    expr
                    | APPEND expr
                    | '|'    expr
                    ;
expr_list_opt    : /* empty */
                    | expr_list
                    ;
expr_list        : expr
                    | multiple_expr_list
                    ;
multiple_expr_list : expr ',' newline_opt expr
                    | multiple_expr_list ',' newline_opt expr
```

```
6638                    ;
6639        expr_opt    : /* empty */
6640                    | expr
6641                    ;
6642        expr        : unary_expr
6643                    | non_unary_expr
6644                    ;
6645        unary_expr  : '+' expr
6646                    | '-' expr
6647                    | unary_expr '^'      expr
6648                    | unary_expr '*'      expr
6649                    | unary_expr '/'      expr
6650                    | unary_expr '%'      expr
6651                    | unary_expr '+'      expr
6652                    | unary_expr '-'      expr
6653                    | unary_expr          non_unary_expr
6654                    | unary_expr '<'      expr
6655                    | unary_expr LE       expr
6656                    | unary_expr NE       expr
6657                    | unary_expr EQ       expr
6658                    | unary_expr '>'      expr
6659                    | unary_expr GE       expr
6660                    | unary_expr '~'      expr
6661                    | unary_expr NO_MATCH expr
6662                    | unary_expr In NAME
6663                    | unary_expr AND newline_opt expr
6664                    | unary_expr OR  newline_opt expr
6665                    | unary_expr '?' expr ':' expr
6666                    | unary_input_function
6667                    ;
6668        non_unary_expr  : '(' expr ')'
6669                        | '!' expr
6670                        | non_unary_expr '^'      expr
6671                        | non_unary_expr '*'      expr
6672                        | non_unary_expr '/'      expr
6673                        | non_unary_expr '%'      expr
6674                        | non_unary_expr '+'      expr
6675                        | non_unary_expr '-'      expr
6676                        | non_unary_expr          non_unary_expr
6677                        | non_unary_expr '<'      expr
6678                        | non_unary_expr LE       expr
6679                        | non_unary_expr NE       expr
6680                        | non_unary_expr EQ       expr
6681                        | non_unary_expr '>'      expr
6682                        | non_unary_expr GE       expr
6683                        | non_unary_expr '~'      expr
6684                        | non_unary_expr NO_MATCH expr
6685                        | non_unary_expr In NAME
6686                        | '(' multiple_expr_list ')' In NAME
6687                        | non_unary_expr AND newline_opt expr
```

```
                        | non_unary_expr OR  newline_opt expr
                        | non_unary_expr '?' expr ':' expr
                        | NUMBER
                        | STRING
                        | lvalue
                        | ERE
                        | lvalue INCR
                        | lvalue DECR
                        | INCR lvalue
                        | DECR lvalue
                        | lvalue POW_ASSIGN expr
                        | lvalue MOD_ASSIGN expr
                        | lvalue MUL_ASSIGN expr
                        | lvalue DIV_ASSIGN expr
                        | lvalue ADD_ASSIGN expr
                        | lvalue SUB_ASSIGN expr
                        | lvalue '=' expr
                        | FUNC_NAME '(' expr_list_opt ')'
                            /* no white space allowed before '(' */
                        | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
                        | BUILTIN_FUNC_NAME
                        | non_unary_input_function
                        ;
        print_expr_list_opt : /* empty */
                        | print_expr_list
                        ;
        print_expr_list  : print_expr
                        | print_expr_list ',' newline_opt print_expr
                        ;
        print_expr       : unary_print_expr
                        | non_unary_print_expr
                        ;
        unary_print_expr : '+' print_expr
                        | '-' print_expr
                        | unary_print_expr '^'      print_expr
                        | unary_print_expr '*'      print_expr
                        | unary_print_expr '/'      print_expr
                        | unary_print_expr '%'      print_expr
                        | unary_print_expr '+'      print_expr
                        | unary_print_expr '-'      print_expr
                        | unary_print_expr          non_unary_print_expr
                        | unary_print_expr '~'      print_expr
                        | unary_print_expr NO_MATCH print_expr
                        | unary_print_expr In NAME
                        | unary_print_expr AND newline_opt print_expr
                        | unary_print_expr OR  newline_opt print_expr
                        | unary_print_expr '?' print_expr ':' print_expr
                        ;
        non_unary_print_expr : '(' expr ')'
                        | '!' print_expr
```

```
6738                        | non_unary_print_expr '^'      print_expr
6739                        | non_unary_print_expr '*'      print_expr
6740                        | non_unary_print_expr '/'      print_expr
6741                        | non_unary_print_expr '%'      print_expr
6742                        | non_unary_print_expr '+'      print_expr
6743                        | non_unary_print_expr '-'      print_expr
6744                        | non_unary_print_expr          non_unary_print_expr
6745                        | non_unary_print_expr '~'      print_expr
6746                        | non_unary_print_expr NO_MATCH print_expr
6747                        | non_unary_print_expr In NAME
6748                        | '(' multiple_expr_list ')' In NAME
6749                        | non_unary_print_expr AND newline_opt print_expr
6750                        | non_unary_print_expr OR  newline_opt print_expr
6751                        | non_unary_print_expr '?' print_expr ':' print_expr
6752                        | NUMBER
6753                        | STRING
6754                        | lvalue
6755                        | ERE
6756                        | lvalue INCR
6757                        | lvalue DECR
6758                        | INCR lvalue
6759                        | DECR lvalue
6760                        | lvalue POW_ASSIGN print_expr
6761                        | lvalue MOD_ASSIGN print_expr
6762                        | lvalue MUL_ASSIGN print_expr
6763                        | lvalue DIV_ASSIGN print_expr
6764                        | lvalue ADD_ASSIGN print_expr
6765                        | lvalue SUB_ASSIGN print_expr
6766                        | lvalue '=' print_expr
6767                        | FUNC_NAME '(' expr_list_opt ')'
6768                            /* no white space allowed before '(' */
6769                        | BUILTIN_FUNC_NAME '(' expr_list_opt ')'
6770                        | BUILTIN_FUNC_NAME
6771                        ;
6772        lvalue          : NAME
6773                        | NAME '[' expr_list ']'
6774                        | '$' expr
6775                        ;
6776        non_unary_input_function : simple_get
6777                        | simple_get '<' expr
6778                        | non_unary_expr '|' simple_get
6779                        ;
6780        unary_input_function : unary_expr '|' simple_get
6781                        ;
6782        simple_get      : GETLINE
6783                        | GETLINE lvalue
6784                        ;
6785        newline_opt     : /* empty */
6786                        | newline_opt NEWLINE
6787                        ;
```

This grammar has several ambiguities that shall be resolved as follows:

- Operator precedence and associativity shall be as described in Table 4-1 (on page 2358).
- In case of ambiguity, an **else** shall be associated with the most immediately preceding **if** that would satisfy the grammar.
- In some contexts, a slash (`'/'`) that is used to surround an ERE could also be the division operator. This shall be resolved in such a way that wherever the division operator could appear, a slash is assumed to be the division operator. (There is no unary division operator.)

One convention that might not be obvious from the formal grammar is where <newline>s are acceptable. There are several obvious placements such as terminating a statement, and a backslash can be used to escape <newline>s between any lexical tokens. In addition, <newline>s without backslashes can follow a comma, an open brace, logical AND operator ("&&"), logical OR operator ("||"), the **do** keyword, the **else** keyword, and the closing parenthesis of an **if**, **for**, or **while** statement. For example:

```
{ print  $1,
         $2 }
```

Lexical Conventions

The lexical conventions for *awk* programs, with respect to the preceding grammar, shall be as follows:

1. Except as noted, *awk* shall recognize the longest possible token or delimiter beginning at a given point.
2. A comment shall consist of any characters beginning with the number sign character and terminated by, but excluding the next occurrence of, a <newline>. Comments shall have no effect, except to delimit lexical tokens.
3. The <newline> shall be recognized as the token **NEWLINE**.
4. A backslash character immediately followed by a <newline> shall have no effect.
5. The token **STRING** shall represent a string constant. A string constant shall begin with the character `'"'`. Within a string constant, a backslash character shall be considered to begin an escape sequence as specified in the table in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`). In addition, the escape sequences in Table 4-2 (on page 2364) shall be recognized. A <newline> shall not occur within a string constant. A string constant shall be terminated by the first unescaped occurrence of the character `'"'` after the one that begins the string constant. The value of the string shall be the sequence of all unescaped characters and values of escape sequences between, but not including, the two delimiting `'"'` characters.
6. The token **ERE** represents an extended regular expression constant. An ERE constant shall begin with the slash character. Within an ERE constant, a backslash character shall be considered to begin an escape sequence as specified in the table in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation. In addition, the escape sequences in Table 4-2 (on page 2364) shall be recognized. The application shall ensure that a <newline> does not occur within an ERE constant. An ERE constant shall be terminated by the first unescaped occurrence of the slash character after the one that begins the ERE constant. The extended regular expression represented by the ERE constant shall be the sequence of all unescaped characters and values of escape sequences between, but not including, the two delimiting slash characters.

7. A <blank> shall have no effect, except to delimit lexical tokens or within **STRING** or **ERE** tokens.

8. The token **NUMBER** shall represent a numeric constant. Its form and numeric value shall be equivalent to either of the tokens **floating-constant** or **integer-constant** as specified by the ISO C standard, with the following exceptions:

   a. An integer constant cannot begin with 0x or include the hexadecimal digits `'a'`, `'b'`, `'c'`, `'d'`, `'e'`, `'f'`, `'A'`, `'B'`, `'C'`, `'D'`, `'E'`, or `'F'`.

   b. The value of an integer constant beginning with 0 shall be taken in decimal rather than octal.

   c. An integer constant cannot include a suffix (`'u'`, `'U'`, `'l'`, or `'L'`).

   d. A floating constant cannot include a suffix (`'f'`, `'F'`, `'l'`, or `'L'`).

   If the value is too large or too small to be representable (see Section 1.7.2 (on page 2207)), the behavior is undefined.

9. A sequence of underscores, digits, and alphabetics from the portable character set (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set), beginning with an underscore or alphabetic, shall be considered a word.

10. The following words are keywords that shall be recognized as individual tokens; the name of the token is the same as the keyword:

BEGIN	**delete**	**END**	**function**	**in**	**printf**
break	**do**	**exit**	**getline**	**next**	**return**
continue	**else**	**for**	**if**	**print**	**while**

11. The following words are names of built-in functions and shall be recognized as the token **BUILTIN_FUNC_NAME**:

atan2	**gsub**	**log**	**split**	**sub**	**toupper**
close	**index**	**match**	**sprintf**	**substr**	
cos	**int**	**rand**	**sqrt**	**system**	
exp	**length**	**sin**	**srand**	**tolower**	

    The above-listed keywords and names of built-in functions are considered reserved words.

12. The token **NAME** shall consist of a word that is not a keyword or a name of a built-in function and is not followed immediately (without any delimiters) by the `'('` character.

13. The token **FUNC_NAME** shall consist of a word that is not a keyword or a name of a built-in function, followed immediately (without any delimiters) by the `'('` character. The `'('` character shall not be included as part of the token.

14. The following two-character sequences shall be recognized as the named tokens:

Token Name	Sequence	Token Name	Sequence
ADD_ASSIGN	+=	**NO_MATCH**	!~
SUB_ASSIGN	–=	**EQ**	==
MUL_ASSIGN	*=	**LE**	<=
DIV_ASSIGN	/=	**GE**	>=
MOD_ASSIGN	%=	**NE**	!=
POW_ASSIGN	^=	**INCR**	++
OR	||	**DECR**	––
AND	&&	**APPEND**	>>

15. The following single characters shall be recognized as tokens whose names are the character:

    ```
    <newline> { } ( ) [ ] , ; + – * % ^ ! > < | ? : ~ $ =
    ```

There is a lexical ambiguity between the token **ERE** and the tokens `'//'` and **DIV_ASSIGN**. When an input sequence begins with a slash character in any syntactic context where the token `'//'` or **DIV_ASSIGN** could appear as the next token in a valid program, the longer of those two tokens that can be recognized shall be recognized. In any other syntactic context where the token **ERE** could appear as the next token in a valid program, the token **ERE** shall be recognized.

EXIT STATUS

The following exit values shall be returned:

0 All input files were processed successfully.

>0 An error occurred.

The exit status can be altered within the program by using an **exit** expression.

CONSEQUENCES OF ERRORS

If any *file* operand is specified and the named file cannot be accessed, *awk* shall write a diagnostic message to standard error and terminate without any further action.

If the program specified by either the *program* operand or a *progfile* operand is not a valid *awk* program (as specified in the EXTENDED DESCRIPTION section), the behavior is undefined.

APPLICATION USAGE

The **index**, **length**, **match**, and **substr** functions should not be confused with similar functions in the ISO C standard; the *awk* versions deal with characters, while the ISO C standard deals with bytes.

Because the concatenation operation is represented by adjacent expressions rather than an explicit operator, it is often necessary to use parentheses to enforce the proper evaluation precedence.

EXAMPLES

The *awk* program specified in the command line is most easily specified within single-quotes (for example, *'program'*) for applications using *sh*, because *awk* programs commonly contain characters that are special to the shell, including double-quotes. In the cases where an *awk* program contains single-quote characters, it is usually easiest to specify most of the program as strings within single-quotes concatenated by the shell with quoted single-quote characters. For example:

```
awk '//'\''/ { print "quote:", $0 }'
```

prints all lines from the standard input containing a single-quote character, prefixed with *quote:*.

The following are examples of simple *awk* programs:

1. Write to the standard output all input lines for which field 3 is greater than 5:

   ```
   $3 > 5
   ```

2. Write every tenth line:

   ```
   (NR % 10) == 0
   ```

3. Write any line with a substring matching the regular expression:

   ```
   /(G|D)(2[0–9][[:alpha:]]*)/
   ```

4. Print any line with a substring containing a 'G' or 'D', followed by a sequence of digits and characters. This example uses character classes **digit** and **alpha** to match language-independent digit and alphabetic characters respectively:

```
/(G|D)([[:digit:][:alpha:]]*)/
```

5. Write any line in which the second field matches the regular expression and the fourth field does not:

```
$2 ~ /xyz/ && $4 !~ /xyz/
```

6. Write any line in which the second field contains a backslash:

```
$2 ~ /\\/
```

7. Write any line in which the second field contains a backslash. Note that backslash escapes are interpreted twice, once in lexical processing of the string and once in processing the regular expression:

```
$2 ~ "\\\\"
```

8. Write the second to the last and the last field in each line. Separate the fields by a colon:

```
{OFS=":";print $(NF-1), $NF}
```

9. Write the line number and number of fields in each line. The three strings representing the line number, the colon, and the number of fields are concatenated and that string is written to standard output:

```
{print NR ":" NF}
```

10. Write lines longer than 72 characters:

```
length($0) > 72
```

11. Write first two fields in opposite order separated by the **OFS**:

```
{ print $2, $1 }
```

12. Same, with input fields separated by comma or <space>s and <tab>s, or both:

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
      { print $2, $1 }
```

13. Add up first column, print sum, and average:

```
      {s += $1 }
END   {print "sum is ", s, " average is", s/NR}
```

14. Write fields in reverse order, one per line (many lines out for each line in):

```
{ for (i = NF; i > 0; --i) print $i }
```

15. Write all lines between occurrences of the strings **start** and **stop**:

```
/start/, /stop/
```

16. Write all lines whose first field is different from the previous one:

```
$1 != prev { print; prev = $1 }
```

17. Simulate *echo*:

```
BEGIN  {
        for (i = 1; i < ARGC; ++i)
        printf("%s%s", ARGV[i], i==ARGC-1?"\n":" ")
```

```
}
```

18. Write the path prefixes contained in the *PATH* environment variable, one per line:

```
BEGIN  {
        n = split (ENVIRON["PATH"], path, ":")
        for (i = 1; i <= n; ++i)
        print path[i]
}
```

19. If there is a file named **input** containing page headers of the form:

    Page #

    and a file named **program** that contains:

```
/Page/   { $2 = n++; }
         { print }
```

    then the command line:

```
awk –f program n=5 input
```

    prints the file **input**, filling in page numbers starting at 5.

RATIONALE

The ISO POSIX-2 standard description is based on the new *awk*, ''nawk'', (see the referenced *The AWK Programming Language*), which introduced a number of new features to the historical *awk*:

1. New keywords: **delete**, **do**, **functin**, **return**
2. New built-in functions: **atan2**, **close**, **cos**, **gsub**, **match**, **rand**, **sin**, **srand**, **sub**, **system**
3. New predefined variables: **FNR**, **ARGC**, **ARGV**, **RSTART**, **RLENGTH**, **SUBSEP**
4. New expression operators: **?**, **:**, **„**, **^**
5. The **FS** variable and the third argument to **split**, now treated as extended regular expressions.
6. The operator precedence, changed to more closely match the C language. Two examples of code that operate differently are:

```
while ( n /= 10 > 1) ...
if (!"wk" ~ /bwk/) ...
```

Several features have been added based on newer implementations of *awk:*

- Multiple instances of **–f** *progfile* are permitted
- The new option **–v** *assignment*
- The new predefined variable **ENVIRON**
- New built-in functions **toupper**, and **tolower**
- More formatting capabilities are added to **printf** to match the ISO C standard

The overall *awk* syntax has always been based on the C language, with a few features from the shell command language and other sources. Because of this, it is not completely compatible with any other language, which has caused confusion for some users. It is not the intent of the standard developers to address such issues. IEEE Std 1003.1-200x has made a few relatively minor changes toward making the language more compatible with the C language as specified by the ISO C standard; most of these changes are based on similar changes in recent

implementations, as described above. There remain several C-language conventions that are not in *awk*. One of the notable ones is the comma operator, which is commonly used to specify multiple expressions in the C language **for** statement. Also, there are various places where *awk* is more restrictive than the C language regarding the type of expression that can be used in a given context. These limitations are due to the different features that the *awk* language does provide.

Regular expressions in *awk* have been extended somewhat from historical implementations to make them a pure superset of extended regular expressions, as defined by IEEE Std 1003.1-200x (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.4, Extended Regular Expressions). The main extensions are internationalization features and interval expressions. Historical implementations of *awk* have long supported backslash escape sequences as an extension to extended regular expressions, and this extension has been retained despite inconsistency with other utilities. The number of escape sequences recognized in both extended regular expressions and strings has varied (generally increasing with time) among implementations. The set specified by IEEE Std 1003.1-200x includes most sequences known to be supported by popular implementations and by the ISO C standard. One sequence that is not supported is hexadecimal value escapes beginning with `'\x'`. This would allow values expressed in more than 9 bits to be used within *awk* as in the ISO C standard. However, because this syntax has a non-deterministic length, it does not permit the subsequent character to be a hexadecimal digit. This limitation can be dealt with in the C language by the use of lexical string concatenation. In the *awk* language, concatenation could also be a solution for strings, but not for extended regular expressions (either lexical ERE tokens or strings used dynamically as regular expressions). Because of this limitation, the feature has not been added to IEEE Std 1003.1-200x.

When a string variable is used in a context where an extended regular expression normally appears (where the lexical token ERE is used in the grammar) the string does not contain the literal slashes.

Some versions of *awk* allow the form:

```
func name(args, ... ) { statements }
```

This has been deprecated by the authors of the language, who asked that it not be included in IEEE Std 1003.1-200x.

Historical implementations of *awk* produce an error if a **next** statement is executed in a **BEGIN** action, and cause *awk* to terminate if a **next** statement is executed in an **END** action. This behavior has not been documented, and it was not believed that it was necessary to standardize it.

The specification of conversions between string and numeric values is much more detailed than in the documentation of historical implementations or in the referenced *The AWK Programming Language*. Although most of the behavior is designed to be intuitive, the details are necessary to ensure compatible behavior from different implementations. This is especially important in relational expressions since the types of the operands determine whether a string or numeric comparison is performed. From the perspective of an application writer, it is usually sufficient to expect intuitive behavior and to force conversions (by adding zero or concatenating a null string) when the type of an expression does not obviously match what is needed. The intent has been to specify historical practice in almost all cases. The one exception is that, in historical implementations, variables and constants maintain both string and numeric values after their original value is converted by any use. This means that referencing a variable or constant can have unexpected side effects. For example, with historical implementations the following program:

```
{
    a = "+2"
```

```
        b = 2
        if (NR % 2)
            c = a + b
        if (a == b)
            print "numeric comparison"
        else
            print "string comparison"
    }
```

would perform a numeric comparison (and output numeric comparison) for each odd-numbered line, but perform a string comparison (and output string comparison) for each even-numbered line. IEEE Std 1003.1-200x ensures that comparisons will be numeric if necessary. With historical implementations, the following program:

```
BEGIN {
    OFMT = "%e"
    print 3.14
    OFMT = "%f"
    print 3.14
}
```

would output `"3.140000e+00"` twice, because in the second **print** statement the constant `"3.14"` would have a string value from the previous conversion. IEEE Std 1003.1-200x requires that the output of the second **print** statement be `"3.140000"`. The behavior of historical implementations was seen as too unintuitive and unpredictable.

It was pointed out that with the rules contained in early drafts, the following script would print nothing:

```
BEGIN {
    y[1.5] = 1
    OFMT = "%e"
    print y[1.5]
}
```

Therefore, a new variable, **CONVFMT**, was introduced. The **OFMT** variable is now restricted to affecting output conversions of numbers to strings and **CONVFMT** is used for internal conversions, such as comparisons or array indexing. The default value is the same as that for **OFMT**, so unless a program changes **CONVFMT** (which no historical program would do), it will receive the historical behavior associated with internal string conversions.

The POSIX *awk* lexical and syntactic conventions are specified more formally than in other sources. Again the intent has been to specify historical practice. One convention that may not be obvious from the formal grammar as in other verbal descriptions is where <newline>s are acceptable. There are several obvious placements such as terminating a statement, and a backslash can be used to escape <newline>s between any lexical tokens. In addition, <newline>s without backslashes can follow a comma, an open brace, a logical AND operator ("&&"), a logical OR operator ("||"), the **do** keyword, the **else** keyword, and the closing parenthesis of an **if**, **for**, or **while** statement. For example:

```
{ print $1,
        $2 }
```

The requirement that *awk* add a trailing <newline> to the program argument text is to simplify the grammar, making it match a text file in form. There is no way for an application or test suite to determine whether a literal <newline> is added or whether *awk* simply acts as if it did.

IEEE Std 1003.1-200x requires several changes from historical implementations in order to support internationalization. Probably the most subtle of these is the use of the decimal-point character, defined by the *LC_NUMERIC* category of the locale, in representations of floating-point numbers. This locale-specific character is used in recognizing numeric input, in converting between strings and numeric values, and in formatting output. However, regardless of locale, the period character (the decimal-point character of the POSIX locale) is the decimal-point character recognized in processing *awk* programs (including assignments in command line arguments). This is essentially the same convention as the one used in the ISO C standard. The difference is that the C language includes the *setlocale*( ) function, which permits an application to modify its locale. Because of this capability, a C application begins executing with its locale set to the C locale, and only executes in the environment-specified locale after an explicit call to *setlocale*( ). However, adding such an elaborate new feature to the *awk* language was seen as inappropriate for IEEE Std 1003.1-200x. It is possible to execute an *awk* program explicitly in any desired locale by setting the environment in the shell.

The undefined behavior resulting from NULs in extended regular expressions allows future extensions for the GNU *gawk* program to process binary data.

The behavior in the case of invalid *awk* programs (including lexical, syntactic, and semantic errors) is undefined because it was considered overly limiting on implementations to specify. In most cases such errors can be expected to produce a diagnostic and a non-zero exit status. However, some implementations may choose to extend the language in ways that make use of certain invalid constructs. Other invalid constructs might be deemed worthy of a warning, but otherwise cause some reasonable behavior. Still other constructs may be very difficult to detect in some implementations. Also, different implementations might detect a given error during an initial parsing of the program (before reading any input files) while others might detect it when executing the program after reading some input. Implementors should be aware that diagnosing errors as early as possible and producing useful diagnostics can ease debugging of applications, and thus make an implementation more usable.

The unspecified behavior from using multi-character **RS** values is to allow possible future extensions based on extended regular expressions used for record separators. Historical implementations take the first character of the string and ignore the others.

Unspecified behavior when *split*(*string*,*array*,<null>) is used is to allow a proposed future extension that would split up a string into an array of individual characters.

In the context of the **getline** function, equally good arguments for different precedences of the **|** and **<** operators can be made. Historical practice has been that:

```
getline < "a" "b"
```

is parsed as:

```
( getline < "a" ) "b"
```

although many would argue that the intent was that the file **ab** should be read. However:

```
getline < "x" + 1
```

parses as:

```
getline < ( "x" + 1 )
```

Similar problems occur with the **|** version of **getline**, particularly in combination with **$**. For example:

```
$"echo hi" | getline
```

(This situation is particularly problematic when used in a **print** statement, where the **|getline** part might be a redirection of the **print**.)

Since in most cases such constructs are not (or at least should not) be used (because they have a natural ambiguity for which there is no conventional parsing), the meaning of these constructs has been made explicitly unspecified. (The effect is that a conforming application that runs into the problem must parenthesize to resolve the ambiguity.) There appeared to be few if any actual uses of such constructs.

Grammars can be written that would cause an error under these circumstances. Where backwards compatibility is not a large consideration, implementors may wish to use such grammars.

Some historical implementations have allowed some built-in functions to be called without an argument list, the result being a default argument list chosen in some ‘‘reasonable’’ way. Use of **length** as a synonym for **length($0)** is the only one of these forms that is thought to be widely known or widely used; this particular form is documented in various places (for example, most historical *awk* reference pages, although not in the referenced *The AWK Programming Language*) as legitimate practice. With this exception, default argument lists have always been undocumented and vaguely defined, and it is not at all clear how (or if) they should be generalized to user-defined functions. They add no useful functionality and preclude possible future extensions that might need to name functions without calling them. Not standardizing them seems the simplest course. The standard developers considered that **length** merited special treatment, however, since it has been documented in the past and sees possibly substantial use in historical programs. Accordingly, this usage has been made legitimate, but Issue 5 removed the obsolescent marking for XSI-conforming implementations and many otherwise conforming applications depend on this feature.

In **sub** and **gsub**, if *repl* is a string literal (the lexical token **STRING**), then two consecutive backslash characters should be used in the string to ensure a single backslash will precede the ampersand when the resultant string is passed to the function. (For example, to specify one literal ampersand in the replacement string, use **gsub**(**ERE**, `"\\&"`).)

Historically the only special character in the *repl* argument of **sub** and **gsub** string functions was the ampersand (`'&'`) character and preceding it with the backslash character was used to turn off its special meaning.

The description in the ISO POSIX-2: 1993 standard introduced behavior such that the backslash character was another special character and it was unspecified whether there were any other special characters. This description introduced several portability problems, some of which are described below, and so it has been replaced with the more historical description. Some of the problems include:

- Historically, to create the replacement string, a script could use **gsub**(**ERE**, `"\\&"`), but with the ISO POSIX-2: 1993 standard wording, it was necessary to use **gsub**(**ERE**, `"\\\\&"`). Backslash characters are doubled here because all string literals are subject to lexical analysis, which would reduce each pair of backslash characters to a single backslash before being passed to **gsub**.
- Since it was unspecified what the special characters were, for portable scripts to guarantee that characters are printed literally, each character had to be preceded with a backslash. (For example, a portable script had to use **gsub**(**ERE**, `"\\h\\i"`) to produce a replacement string of `"hi"`.)

The description for comparisons in the ISO POSIX-2: 1993 standard did not properly describe historical practice because of the way numeric strings are compared as numbers. The current rules cause the following code:

```
if (0 == "000")
    print "strange, but true"
else
    print "not true"
```

to do a numeric comparison, causing the **if** to succeed. It should be intuitively obvious that this is incorrect behavior, and indeed, no historical implementation of *awk* actually behaves this way.

To fix this problem, the definition of *numeric string* was enhanced to include only those values obtained from specific circumstances (mostly external sources) where it is not possible to determine unambiguously whether the value is intended to be a string or a numeric.

Variables that are assigned to a numeric string shall also be treated as a numeric string. (For example, the notion of a numeric string can be propagated across assignments.) In comparisons, all variables having the uninitialized value are to be treated as a numeric operand evaluating to the numeric value zero.

Uninitialized variables include all types of variables including scalars, array elements, and fields. The definition of an uninitialized value in **Variables and Special Variables** (on page 2361) is necessary to describe the value placed on uninitialized variables and on fields that are valid (for example, **< $NF**) but have no characters in them and to describe how these variables are to be used in comparisons. A valid field, such as **$1**, that has no characters in it can be obtained by from an input line of `"\t\t"` when **FS='\t'**. Historically, the comparison (**$1**<10) was done numerically after evaluating **$1** to the value zero.

The phrase ‘‘... also shall have the numeric value of the numeric string’’ was removed from several sections of the ISO POSIX-2:1993 standard because is specifies an unnecessary implementation detail. It is not necessary for IEEE Std 1003.1-200x to specify that these objects be assigned two different values. It is only necessary to specify that these objects may evaluate to two different values depending on context.

The description of numeric string processing is based on the behavior of the *atof*( ) function in the ISO C standard. While it is not a requirement for an implementation to use this function, many historical implementations of *awk* do. In the ISO C standard, floating-point constants use a period as a decimal point character for the language itself, independent of the current locale, but the *atof*( ) function and the associated *strtod*( ) function use the decimal point character of the current locale when converting strings to numeric values. Similarly in *awk*, floating-point constants in an *awk* script use a period independent of the locale, but input strings use the decimal point character of the locale.

FUTURE DIRECTIONS

None.

SEE ALSO

grep, *lex*, *sed*, the System Interfaces volume of IEEE Std 1003.1-200x, *atof*( ), *setlocale*( ), *strtod*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

FUTURE DIRECTIONS section added.

Issue 6

The *awk* utility is aligned with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements. |

IEEE PASC Interpretation 1003.2 #211 is applied, adding the sentence ‘‘An occurrence of two consecutive backslashes shall be interpreted as just a single literal backslash character.’’ into the description of the **sub** string function.

NAME

basename — return non-directory portion of a pathname

SYNOPSIS

```
basename string [suffix]
```

DESCRIPTION

The *string* operand shall be treated as a pathname, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.266, Pathname. The string *string* shall be converted to the filename corresponding to the last pathname component in *string* and then the suffix string *suffix*, if present, shall be removed. This shall be done by performing actions equivalent to the following steps in order:

1. If *string* is a null string, it is unspecified whether the resulting string is `'.'` or a null string. In either case, skip steps 2 through 6.
2. If *string* is `"//"`, it is implementation-defined whether steps 3 to 6 are skipped or processed.
3. If *string* consists entirely of slash characters, *string* shall be set to a single slash character. In this case, skip steps 4 to 6.
4. If there are any trailing slash characters in *string*, they shall be removed.
5. If there are any slash characters remaining in *string*, the prefix of *string* up to and including the last slash character in *string* shall be removed.
6. If the *suffix* operand is present, is not identical to the characters remaining in *string*, and is identical to a suffix of the characters remaining in *string*, the suffix *suffix* shall be removed from *string*. Otherwise, *string* is not modified by this step. It shall not be considered an error if *suffix* is not found in *string*.

The resulting string shall be written to standard output.

OPTIONS

None.

OPERANDS

The following operands shall be supported:

string A string.

suffix A string.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *basename*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The *basename* utility shall write a line to the standard output in the following format:

```
"%s\n", <resulting string>
```

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The definition of *pathname* specifies implementation-defined behavior for pathnames starting with two slash characters. Therefore, applications shall not arbitrarily add slashes to the beginning of a pathname unless they can ensure that there are more or less than two or are prepared to deal with the implementation-defined consequences.

EXAMPLES

If the string *string* is a valid pathname:

```
$(basename "string")
```

produces a filename that could be used to open the file named by *string* in the directory returned by:

```
$(dirname "string")
```

If the string *string* is not a valid pathname, the same algorithm is used, but the result need not be a valid filename. The *basename* utility is not expected to make any judgements about the validity of *string* as a pathname; it just follows the specified algorithm to produce a result string.

The following shell script compiles **/usr/src/cmd/cat.c** and moves the output to a file named **cat** in the current directory when invoked with the argument **/usr/src/cmd/cat** or with the argument **/usr/src/cmd/cat.c**:

```
        c99 $(dirname "$1")/$(basename "$1" .c).c
        mv a.out $(basename "$1" .c)
```

RATIONALE

The behaviors of *basename* and *dirname* have been coordinated so that when *string* is a valid pathname:

```
$(basename "string")
```

would be a valid filename for the file in the directory:

```
$(dirname "string")
```

This would not work for the early proposal versions of these utilities due to the way it specified handling of trailing slashes.

Since the definition of *pathname* specifies implementation-defined behavior for pathnames starting with two slash characters, this volume of IEEE Std 1003.1-200x specifies similar implementation-defined behavior for the *basename* and *dirname* utilities.

FUTURE DIRECTIONS

None.

SEE ALSO

dirname, Section 2.5 (on page 2235)

CHANGE HISTORY

First released in Issue 2.

Issue 6

IEEE PASC Interpretation 1003.2 #164 is applied.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

NAME

batch — schedule commands to be executed in a batch queue

SYNOPSIS

UP

```
batch
```

DESCRIPTION

The *batch* utility shall read commands from standard input and schedule them for execution in a batch queue. It shall be the equivalent of the command:

```
at -q b -m now
```

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 invocation of *batch*.

XSI Users shall be permitted to use *batch* if their name appears in the file **/usr/lib/cron/at.allow**. If that file does not exist, the file **/usr/lib/cron/at.deny** shall be checked to determine whether the user shall be denied access to *batch*. If neither file exists, only a process with the appropriate privileges shall be allowed to submit a job. If only **at.deny** exists and is empty, global usage shall be permitted. The **at.allow** and **at.deny** files shall consist of one user name per line.

OPTIONS

None.

OPERANDS

None.

STDIN

The standard input shall be a text file consisting of commands acceptable to the shell command language described in Chapter 2 (on page 2231).

INPUT FILES

XSI The text files **/usr/lib/cron/at.allow** and **/usr/lib/cron/at.deny** shall contain zero or more user names, one per line, of users who are, respectively, authorized or denied access to the *at* and *batch* utilities.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *batch*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

LC_TIME Determine the format and contents for date and time strings written by *batch*.

7379 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

7380 *SHELL* Determine the name of a command interpreter to be used to invoke the at-job. If
7381 the variable is unset or null, *sh* shall be used. If it is set to a value other than a name
7382 for *sh*, the implementation shall do one of the following: use that shell; use *sh*; use
7383 the login shell from the user database; any of the preceding accompanied by a
7384 warning diagnostic about which was chosen.

7385 *TZ* Determine the timezone. The job shall be submitted for execution at the time
7386 specified by *timespec* or **–t** *time* relative to the timezone specified by the *TZ*
7387 variable. If *timespec* specifies a timezone, it overrides *TZ*. If *timespec* does not
7388 specify a timezone and *TZ* is unset or null, an unspecified default timezone shall
7389 be used.

7390 **ASYNCHRONOUS EVENTS**
7391 Default.

7392 **STDOUT**
7393 When standard input is a terminal, prompts of unspecified format for each line of the user input
7394 described in the STDIN section may be written to standard output.

7395 **STDERR**
7396 The following shall be written to standard error when a job has been successfully submitted:

7397 `"job %s at %s\n", at_job_id, <date>`

7398 where *date* shall be equivalent in format to the output of:

7399 `date +"%a %b %e %T %Y"`

7400 The date and time written shall be adjusted so that they appear in the timezone of the user (as
7401 determined by the *TZ* variable).

7402 Neither this, nor warning messages concerning the selection of the command interpreter, are
7403 considered a diagnostic that changes the exit status.

7404 Diagnostic messages, if any, shall be written to standard error.

7405 **OUTPUT FILES**
7406 None.

7407 **EXTENDED DESCRIPTION**
7408 None.

7409 **EXIT STATUS**
7410 The following exit values shall be returned:

7411 0 Successful completion.

7412 >0 An error occurred.

7413 **CONSEQUENCES OF ERRORS**
7414 The job shall not be scheduled.

APPLICATION USAGE

It may be useful to redirect standard output within the specified commands.

EXAMPLES

1. This sequence can be used at a terminal:

```
batch
sort < file >outfile
EOT
```

2. 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 !
```

RATIONALE

Early proposals described *batch* in a manner totally separated from *at*, even though the historical model treated it almost as a synonym for *at* **–qb**. 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 did not merit the change to the historical interface.

The **–m** option was included on the equivalent *at* command because it is historical practice to mail results to the submitter, even if all job-produced output is redirected. As explained in the RATIONALE for *at*, the **now** keyword submits the job for immediate execution (after scheduling delays), despite some historical systems where *at* **now** would have been considered an error.

FUTURE DIRECTIONS

None.

SEE ALSO

at

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The NAME is changed to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

NAME

bc — arbitrary-precision arithmetic language

SYNOPSIS

```
bc [-l] [file ...]
```

DESCRIPTION

The *bc* utility shall implement an arbitrary precision calculator. It shall take input from any files given, then read from the standard input. If the standard input and standard output to *bc* are attached to a terminal, the invocation of *bc* shall be considered to be *interactive*, causing behavioral constraints described in the following sections.

OPTIONS

The *bc* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following option shall be supported:

–l (The letter ell.) Define the math functions and initialize *scale* to 20, instead of the default zero; see the EXTENDED DESCRIPTION section.

OPERANDS

The following operand shall be supported:

file A pathname of a text file containing *bc* program statements. After all *file*s have been read, *bc* shall read the standard input.

STDIN

See the INPUT FILES section.

INPUT FILES

Input files shall be text files containing a sequence of comments, statements, and function definitions that shall be executed as they are read.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *bc*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The output of the *bc* utility shall be controlled by the program read, and consist of zero or more lines containing the value of all executed expressions without assignments. The radix and precision of the output shall be controlled by the values of the **obase** and **scale** variables; see the EXTENDED DESCRIPTION section.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

Grammar

The grammar in this section and the lexical conventions in the following section shall together describe the syntax for *bc* programs. The general conventions for this style of grammar are described in Section 1.10 (on page 2220). A valid program can be represented as the non-terminal symbol **program** in the grammar. This formal syntax shall take precedence over the text syntax description.

```
%token    EOF NEWLINE STRING LETTER NUMBER

%token    MUL_OP
/*        '*', '/', '%'                              */

%token    ASSIGN_OP
/*        '=', '+=', '-=', '*=', '/=', '%=', '^=' */

%token    REL_OP
/*        '==', '<=', '>=', '!=', '<', '>'         */

%token    INCR_DECR
/*        '++', '--'                                 */

%token    Define    Break    Quit    Length
/*        'define', 'break', 'quit', 'length'      */

%token    Return    For    If    While    Sqrt
/*        'return', 'for', 'if', 'while', 'sqrt'   */

%token    Scale    Ibase    Obase    Auto
/*        'scale', 'ibase', 'obase', 'auto'        */

%start    program

%%

program              : EOF
                     | input_item program
                     ;

input_item           : semicolon_list NEWLINE
                     | function
                     ;

semicolon_list       : /* empty */
                     | statement
                     | semicolon_list ';' statement
                     | semicolon_list ';'
```

```
                        ;

statement_list          : /* empty */
                        | statement
                        | statement_list NEWLINE
                        | statement_list NEWLINE statement
                        | statement_list ';'
                        | statement_list ';' statement
                        ;

statement               : expression
                        | STRING
                        | Break
                        | Quit
                        | Return
                        | Return '(' return_expression ')'
                        | For '(' expression ';'
                            relational_expression ';'
                            expression ')' statement
                        | If '(' relational_expression ')' statement
                        | While '(' relational_expression ')' statement
                        | '{' statement_list '}'
                        ;

function                : Define LETTER '(' opt_parameter_list ')'
                            '{' NEWLINE opt_auto_define_list
                            statement_list '}'
                        ;

opt_parameter_list      : /* empty */
                        | parameter_list
                        ;

parameter_list          : LETTER
                        | define_list ',' LETTER
                        ;

opt_auto_define_list    : /* empty */
                        | Auto define_list NEWLINE
                        | Auto define_list ';'
                        ;

define_list             : LETTER
                        | LETTER '[' ']'
                        | define_list ',' LETTER
                        | define_list ',' LETTER '[' ']'
                        ;

opt_argument_list       : /* empty */
                        | argument_list
                        ;

argument_list           : expression
                        | LETTER '[' ']' ',' argument_list"
                        ;
```

```
relational_expression : expression
                      | expression REL_OP expression
                      ;

return_expression     : /* empty */
                      | expression
                      ;

expression            : named_expression
                      | NUMBER
                      | '(' expression ')'
                      | LETTER '(' opt_argument_list ')'
                      | '-' expression
                      | expression '+' expression
                      | expression '-' expression
                      | expression MUL_OP expression
                      | expression '^' expression
                      | INCR_DECR named_expression
                      | named_expression INCR_DECR
                      | named_expression ASSIGN_OP expression
                      | Length '(' expression ')'
                      | Sqrt '(' expression ')'
                      | Scale '(' expression ')'
                      ;

named_expression      : LETTER
                      | LETTER '[' expression ']'
                      | Scale
                      | Ibase
                      | Obase
                      ;
```

Lexical Conventions in bc

The lexical conventions for *bc* programs, with respect to the preceding grammar, shall be as follows:

1. Except as noted, *bc* shall recognize the longest possible token or delimiter beginning at a given point.
2. A comment shall consist of any characters beginning with the two adjacent characters "/*" and terminated by the next occurrence of the two adjacent characters "*/". Comments shall have no effect except to delimit lexical tokens.
3. The <newline> shall be recognized as the token **NEWLINE**.
4. The token **STRING** shall represent a string constant; it shall consist of any characters beginning with the double-quote character ('"') and terminated by another occurrence of the double-quote character. The value of the string is the sequence of all characters between, but not including, the two double-quote characters. All characters shall be taken literally from the input, and there is no way to specify a string containing a double-quote character. The length of the value of each string shall be limited to {BC_STRING_MAX} bytes.
5. A <blank> shall have no effect except as an ordinary character if it appears within a **STRING** token, or to delimit a lexical token other than **STRING**.

6. The combination of a backslash character immediately followed by a <newline> shall have no effect other than to delimit lexical tokens with the following exceptions:

   - It shall be interpreted as the character sequence "\<newline>" in **STRING** tokens.
   - It shall be ignored as part of a multi-line **NUMBER** token.

7. The token **NUMBER** shall represent a numeric constant. It shall be recognized by the following grammar:

```
NUMBER  : integer
        | '.' integer
        | integer '.'
        | integer '.' integer
        ;

integer : digit
        | integer digit
        ;

digit   : 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
        | 8 | 9 | A | B | C | D | E | F
        ;
```

8. The value of a **NUMBER** token shall be interpreted as a numeral in the base specified by the value of the internal register **ibase** (described below). Each of the **digit** characters shall have the value from 0 to 15 in the order listed here, and the period character shall represent the radix point. The behavior is undefined if digits greater than or equal to the value of **ibase** appear in the token. However, note the exception for single-digit values being assigned to **ibase** and **obase** themselves, in **Operations in bc** (on page 2400).

9. The following keywords shall be recognized as tokens:

auto	**ibase**	**length**	**return**	**while**
break	**if**	**obase**	**scale**	
define	**for**	**quit**	**sqrt**	

10. Any of the following characters occurring anywhere except within a keyword shall be recognized as the token **LETTER**:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

11. The following single-character and two-character sequences shall be recognized as the token **ASSIGN_OP**:

```
=    +=    -=    *=    /=    %=    ^=
```

12. If an '=' character, as the beginning of a token, is followed by a '-' character with no intervening delimiter, the behavior is undefined.

13. The following single-characters shall be recognized as the token **MUL_OP**:

```
*    /    %
```

14. The following single-character and two-character sequences shall be recognized as the token **REL_OP**:

```
==    <=    >=    !=    <    >
```

15. The following two-character sequences shall be recognized as the token **INCR_DECR**:

7661 `++ --`

7662 16. The following single characters shall be recognized as tokens whose names are the
7663 character:

7664 `<newline> ( ) , + - ; [ ] ^ { }`

7665 17. The token **EOF** is returned when the end of input is reached.

7666 **Operations in bc**

7667 There are three kinds of identifiers: ordinary identifiers, array identifiers, and function
7668 identifiers. All three types consist of single lowercase letters. Array identifiers shall be followed
7669 by square brackets ("`[]`"). An array subscript is required except in an argument or auto list.
7670 Arrays are singly dimensioned and can contain up to {BC_DIM_MAX} elements. Indexing shall
7671 begin at zero so an array is indexed from 0 to {BC_DIM_MAX}−1. Subscripts shall be truncated
7672 to integers. The application shall ensure that function identifiers are followed by parentheses,
7673 possibly enclosing arguments. The three types of identifiers do not conflict.

7674 The following table summarizes the rules for precedence and associativity of all operators.
7675 Operators on the same line shall have the same precedence; rows are in order of decreasing
7676 precedence.

7677 **Table 4-3** Operators in *bc*

7678 Operator	Associativity
7679 `++, --`	N/A
7680 `unary -`	N/A
7681 `^`	Right to left
7682 `*, /, %`	Left to right
7683 `+, binary -`	Left to right
7684 `=, +=, -=, *=, /=, %=, ^=`	Right to left
7685 `==, <=, >=, !=, <, >`	None

7686 Each expression or named expression has a *scale*, which is the number of decimal digits that
7687 shall be maintained as the fractional portion of the expression.

7688 *Named expressions* are places where values are stored. Named expressions shall be valid on the
7689 left side of an assignment. The value of a named expression shall be the value stored in the place
7690 named. Simple identifiers and array elements are named expressions; they have an initial value
7691 of zero and an initial scale of zero.

7692 The internal registers **scale**, **ibase**, and **obase** are all named expressions. The scale of an
7693 expression consisting of the name of one of these registers shall be zero; values assigned to any
7694 of these registers are truncated to integers. The **scale** register shall contain a global value used in
7695 computing the scale of expressions (as described below). The value of the register **scale** is
7696 limited to 0 ≤ **scale** ≤ {BC_SCALE_MAX} and shall have a default value of zero. The **ibase** and
7697 **obase** registers are the input and output number radix, respectively. The value of **ibase** shall be
7698 limited to:

7699 `2 ≤ ibase ≤ 16`

7700 The value of **obase** shall be limited to:

7701 `2 ≤ obase ≤ {BC_BASE_MAX}`

7702 When either **ibase** or **obase** is assigned a single **digit** value from the list in **Lexical Conventions**
7703 **in bc** (on page 2398), the value shall be assumed in hexadecimal. (For example, **ibase**=A sets to

base ten, regardless of the current **ibase** value.) Otherwise, the behavior is undefined when digits greater than or equal to the value of **ibase** appear in the input. Both **ibase** and **obase** shall have initial values of 10.

Internal computations shall be conducted as if in decimal, regardless of the input and output bases, to the specified number of decimal digits. When an exact result is not achieved, (for example, **scale**=0; 3.2 / 1) the result shall be truncated.

For all values of **obase** specified by this volume of IEEE Std 1003.1-200x, *bc* shall output numeric values by performing each of the following steps in order:

1. If the value is less than zero, a hyphen (`'-'`) character shall be output.
2. One of the following is output, depending on the numerical value:
   - If the absolute value of the numerical value is greater than or equal to one, the integer portion of the value shall be output as a series of digits appropriate to **obase** (as described below) most significant digit first. The most significant non-zero digit shall be output next, followed by each successively less significant digit.
   - If the absolute value of the numerical value is less than one but greater than zero and the scale of the numerical value is greater than zero, it is unspecified whether the character 0 is output.
   - If the numerical value is zero, the character 0 shall be output.
3. If the scale of the value is greater than zero and the numeric value is not zero, a period character shall be output, followed by a series of digits appropriate to **obase** (as described below) representing the most significant portion of the fractional part of the value. If *s* represents the scale of the value being output, the number of digits output shall be *s* if **obase** is 10, less than or equal to *s* if **obase** is greater than 10, or greater than or equal to *s* if **obase** is less than 10. For **obase** values other than 10, this should be the number of digits needed to represent a precision of 10^s.

For **obase** values from 2 to 16, valid digits are the first **obase** of the single characters:

```
0 1 2 3 4 5 6 7 8 9 A B C D E F
```

which represent the values zero to 15, inclusive, respectively.

For bases greater than 16, each digit shall be written as a separate multi-digit decimal number. Each digit except the most significant fractional digit shall be preceded by a single <space>. For bases from 17 to 100, *bc* shall write two-digit decimal numbers; for bases from 101 to 1 000, three-digit decimal strings, and so on. For example, the decimal number 1 024 in base 25 would be written as:

```
Δ01Δ15Δ24
```

in base 125, as:

```
Δ008Δ024
```

Very large numbers shall be split across lines with 70 characters per line in the POSIX locale; other locales may split at different character boundaries. Lines that are continued shall end with a backslash (`'\'`).

A function call shall consist of a function name followed by parentheses containing a comma-separated list of expressions, which are the function arguments. A whole array passed as an argument shall be specified by the array name followed by empty square brackets. All function arguments shall be passed by value. As a result, changes made to the formal parameters shall have no effect on the actual arguments. If the function terminates by executing a **return**

statement, the value of the function shall be the value of the expression in the parentheses of the **return** statement or shall be zero if no expression is provided or if there is no **return** statement.

The result of **sqrt**(*expression*) shall be the square root of the expression. The result shall be truncated in the least significant decimal place. The scale of the result shall be the scale of the expression or the value of **scale**, whichever is larger.

The result of **length**(*expression*) shall be the total number of significant decimal digits in the expression. The scale of the result shall be zero.

The result of **scale**(*expression*) shall be the scale of the expression. The scale of the result shall be zero.

A numeric constant shall be an expression. The scale shall be the number of digits that follow the radix point in the input representing the constant, or zero if no radix point appears.

The sequence (*expression*) shall be an expression with the same value and scale as *expression*. The parentheses can be used to alter the normal precedence.

The semantics of the unary and binary operators are as follows:

–expression
: The result shall be the negative of the *expression*. The scale of the result shall be the scale of *expression*.

The unary increment and decrement operators shall not modify the scale of the named expression upon which they operate. The scale of the result shall be the scale of that named expression.

++named-expression
: The named expression shall be incremented by one. The result shall be the value of the named expression after incrementing.

––named-expression
: The named expression shall be decremented by one. The result shall be the value of the named expression after decrementing.

named-expression++
: The named expression shall be incremented by one. The result shall be the value of the named expression before incrementing.

named-expression––
: The named expression shall be decremented by one. The result shall be the value of the named expression before decrementing.

The exponentiation operator, circumflex (`'^'`), shall bind right to left.

expression^expression
: The result shall be the first *expression* raised to the power of the second *expression*. If the second expression is not an integer, the behavior is undefined. If *a* is the scale of the left expression and *b* is the absolute value of the right expression, the scale of the result shall be:

```
if b >= 0 min(a * b, max(scale, a)) if b < 0 scale
```

The multiplicative operators (`'*'`, `'/'`, `'%'`) shall bind left to right.

*expression*expression*
: The result shall be the product of the two expressions. If *a* and *b* are the scales of the two expressions, then the scale of the result shall be:

```
min(a+b,max(scale,a,b))
```

expression / *expression*

The result shall be the quotient of the two expressions. The scale of the result shall be the value of **scale**.

expression%*expression*

For expressions *a* and *b*, *a*%*b* shall be evaluated equivalent to the steps:

1. Compute *a* / *b* to current scale.
2. Use the result to compute:

   ```
   a - (a / b) * b
   ```

   to scale:

   ```
   max(scale + scale(b), scale(a))
   ```

The scale of the result shall be:

```
max(scale + scale(b), scale(a))
```

When **scale** is zero, the '%' operator is the mathematical remainder operator.

The additive operators ('+', '−') shall bind left to right.

expression+*expression*

The result shall be the sum of the two expressions. The scale of the result shall be the maximum of the scales of the expressions.

expression−*expression*

The result shall be the difference of the two expressions. The scale of the result shall be the maximum of the scales of the expressions.

The assignment operators ('=', "+=", "−=", "*=", "/=", "%=", "^=") shall bind right to left.

named-expression=*expression*

This expression shall result in assigning the value of the expression on the right to the named expression on the left. The scale of both the named expression and the result shall be the scale of *expression*.

The compound assignment forms:

named-expression <operator>= expression

shall be equivalent to:

named-expression=named-expression <operator> expression

except that the *named-expression* shall be evaluated only once.

Unlike all other operators, the relational operators ('<', '>', "<=", ">=", "==", "!=") shall be only valid as the object of an **if**, **while**, or inside a **for** statement.

expression1<*expression2*

The relation shall be true if the value of *expression1* is strictly less than the value of *expression2*.

expression1>*expression2*

The relation shall be true if the value of *expression1* is strictly greater than the value of *expression2*.

expression1<=*expression2*
The relation shall be true if the value of *expression1* is less than or equal to the value of *expression2*.

expression1>=*expression2*
The relation shall be true if the value of *expression1* is greater than or equal to the value of *expression2*.

expression1= =*expression2*
The relation shall be true if the values of *expression1* and *expression2* are equal.

expression1!=*expression2*
The relation shall be true if the values of *expression1* and *expression2* are unequal.

There are only two storage classes in *bc*, global and automatic (local). Only identifiers that are local to a function need be declared with the **auto** command. The arguments to a function shall be local to the function. All other identifiers are assumed to be global and available to all functions. All identifiers, global and local, have initial values of zero. Identifiers declared as auto shall be allocated on entry to the function and released on returning from the function. They therefore do not retain values between function calls. Auto arrays shall be specified by the array name followed by empty square brackets. On entry to a function, the old values of the names that appear as parameters and as automatic variables shall be pushed onto a stack. Until the function returns, reference to these names shall refer only to the new values.

References to any of these names from other functions that are called from this function also refer to the new value until one of those functions uses the same name for a local variable.

When a statement is an expression, unless the main operator is an assignment, execution of the statement shall write the value of the expression followed by a <newline>.

When a statement is a string, execution of the statement shall write the value of the string.

Statements separated by semicolons or <newline>s shall be executed sequentially. In an interactive invocation of *bc*, each time a <newline> is read that satisfies the grammatical production:

```
input_item : semicolon_list NEWLINE
```

the sequential list of statements making up the **semicolon_list** shall be executed immediately and any output produced by that execution shall be written without any delay due to buffering.

In an **if** statement (**if**(*relation*) *statement*), the *statement* shall be executed if the relation is true.

The **while** statement (**while**(*relation*) *statement*) implements a loop in which the *relation* is tested; each time the *relation* is true, the *statement* shall be executed and the *relation* retested. When the *relation* is false, execution shall resume after *statement*.

A **for** statement(**for**(*expression*; *relation*; *expression*) *statement*) shall be the same as:

```
first-expression
while (relation) {
    statement
    last-expression
}
```

The application shall ensure that all three expressions are present.

The **break** statement shall cause termination of a **for** or **while** statement.

The **auto** statement (**auto** *identifier* **[**,*identifier***]** ...) shall cause the values of the identifiers to be pushed down. The identifiers can be ordinary identifiers or array identifiers. Array identifiers

shall be specified by following the array name by empty square brackets. The application shall ensure that the **auto** statement is the first statement in a function definition.

A **define** statement:

```
define LETTER ( opt_parameter_list ) {
    opt_auto_define_list
    statement_list
}
```

defines a function named **LETTER**. If a function named **LETTER** was previously defined, the **define** statement shall replace the previous definition. The expression:

```
LETTER ( opt_argument_list )
```

shall invoke the function named **LETTER**. The behavior is undefined if the number of arguments in the invocation does not match the number of parameters in the definition. Functions shall be defined before they are invoked. A function shall be considered to be defined within its own body, so recursive calls are valid. The values of numeric constants within a function shall be interpreted in the base specified by the value of the **ibase** register when the function is invoked.

The **return** statements (**return** and **return**(*expression*)) shall cause termination of a function, popping of its auto variables, and specification of the result of the function. The first form shall be equivalent to **return**(0). The value and scale of the result returned by the function shall be the value and scale of the expression returned.

The **quit** statement (**quit**) shall stop execution of a *bc* program at the point where the statement occurs in the input, even if it occurs in a function definition, or in an **if**, **for**, or **while** statement.

The following functions shall be defined when the **–l** option is specified:

s(*expression*)
: Sine of argument in radians.

c(*expression*)
: Cosine of argument in radians.

a(*expression*)
: Arctangent of argument.

l(*expression*)
: Natural logarithm of argument.

e(*expression*)
: Exponential function of argument.

j(*expression*, *expression*)
: Bessel function of integer order.

The scale of the result returned by these functions shall be the value of the **scale** register at the time the function is invoked. The value of the **scale** register after these functions have completed their execution shall be the same value it had upon invocation. The behavior is undefined if any of these functions is invoked with an argument outside the domain of the mathematical function.

EXIT STATUS

The following exit values shall be returned:

0 All input files were processed successfully.

unspecified An error occurred.

CONSEQUENCES OF ERRORS

If any *file* operand is specified and the named file cannot be accessed, *bc* shall write a diagnostic message to standard error and terminate without any further action.

In an interactive invocation of *bc*, the utility should print an error message and recover following any error in the input. In a non-interactive invocation of *bc*, invalid input causes undefined behavior.

APPLICATION USAGE

Automatic variables in *bc* do not work in exactly the same way as in either C or PL/1.

For historical reasons, the exit status from *bc* cannot be relied upon to indicate that an error has occurred. Returning zero after an error is possible. Therefore, *bc* should be used primarily by interactive users (who can react to error messages) or by application programs that can somehow validate the answers returned as not including error messages.

The *bc* utility always uses the period (`'.'`) character to represent a radix point, regardless of any decimal-point character specified as part of the current locale. In languages like C or *awk*, the period character is used in program source, so it can be portable and unambiguous, while the locale-specific character is used in input and output. Because there is no distinction between source and input in *bc*, this arrangement would not be possible. Using the locale-specific character in *bc*'s input would introduce ambiguities into the language; consider the following example in a locale with a comma as the decimal-point character:

```
define f(a,b) {
    ...
}
...
f(1,2,3)
```

Because of such ambiguities, the period character is used in input. Having input follow different conventions from output would be confusing in either pipeline usage or interactive usage, so the period is also used in output.

EXAMPLES

In the shell, the following assigns an approximation of the first ten digits of `'π'` to the variable *x*:

```
x=$(printf "%s\n" 'scale = 10; 104348/33215' | bc)
```

The following *bc* program prints the same approximation of `'π'`, with a label, to standard output:

```
scale = 10
"pi equals "
104348 / 33215
```

The following defines a function to compute an approximate value of the exponential function (note that such a function is predefined if the **–l** option is specified):

```
scale = 20
define e(x){
    auto a, b, c, i, s
    a = 1
    b = 1
    s = 1
```

```
        for (i = 1; 1 == 1; i++){
            a = a*x
            b = b*i
            c = a/b
            if (c == 0) {
                 return(s)
            }
            s = s+c
        }
    }
```

The following prints approximate values of the exponential function of the first ten integers:

```
    for (i = 1; i <= 10; ++i) {
        e(i)
    }
```

RATIONALE

The *bc* utility is implemented historically as a front-end processor for *dc*; *dc* was not selected to be part of this volume of IEEE Std 1003.1-200x because *bc* was thought to have a more intuitive programmatic interface. Current implementations that implement *bc* using *dc* are expected to be compliant.

The exit status for error conditions has been left unspecified for several reasons:

- The *bc* utility is used in both interactive and non-interactive situations. Different exit codes may be appropriate for the two uses.
- It is unclear when a non-zero exit should be given; divide-by-zero, undefined functions, and syntax errors are all possibilities.
- It is not clear what utility the exit status has.
- In the 4.3 BSD, System V, and Ninth Edition implementations, *bc* works in conjunction with *dc*. The *dc* utility is the parent, *bc* is the child. This was done to cleanly terminate *bc* if *dc* aborted.

The decision to have *bc* exit upon encountering an inaccessible input file is based on the belief that *bc file1 file2* is used most often when at least *file1* contains data/function declarations/initializations. Having *bc* continue with prerequisite files missing is probably not useful. There is no implication in the CONSEQUENCES OF ERRORS section that *bc* must check all its files for accessibility before opening any of them.

There was considerable debate on the appropriateness of the language accepted by *bc*. Several reviewers preferred to see either a pure subset of the C language or some changes to make the language more compatible with C. While the *bc* language has some obvious similarities to C, it has never claimed to be compatible with any version of C. An interpreter for a subset of C might be a very worthwhile utility, and it could potentially make *bc* obsolete. However, no such utility is known in historical practice, and it was not within the scope of this volume of IEEE Std 1003.1-200x to define such a language and utility. If and when they are defined, it may be appropriate to include them in a future version of this volume of IEEE Std 1003.1-200x. This left the following alternatives:

1. Exclude any calculator language from this volume of IEEE Std 1003.1-200x.

   The consensus of the standard developers was that a simple programmatic calculator language is very useful for both applications and interactive users. The only arguments for excluding any calculator were that it would become obsolete if and when a C-compatible

one emerged, or that the absence would encourage the development of such a C-compatible one. These arguments did not sufficiently address the needs of current application writers.

2. Standardize the historical *dc*, possibly with minor modifications.

   The consensus of the standard developers was that *dc* is a fundamentally less usable language and that that would be far too severe a penalty for avoiding the issue of being similar to but incompatible with C.

3. Standardize the historical *bc*, possibly with minor modifications.

   This was the approach taken. Most of the proponents of changing the language would not have been satisfied until most or all of the incompatibilities with C were resolved. Since most of the changes considered most desirable would break historical applications and require significant modification to historical implementations, almost no modifications were made. The one significant modification that was made was the replacement of the historical *bc* assignment operators "=+", and so on, with the more modern "+=", and so on. The older versions are considered to be fundamentally flawed because of the lexical ambiguity in uses like *a*=–1.

   In order to permit implementations to deal with backwards compatibility as they see fit, the behavior of this one ambiguous construct was made undefined. (At least three implementations have been known to support this change already, so the degree of change involved should not be great.)

The '%' operator is the mathematical remainder operator when **scale** is zero. The behavior of this operator for other values of **scale** is from historical implementations of *bc*, and has been maintained for the sake of historical applications despite its non-intuitive nature.

Historical implementations permit setting **ibase** and **obase** to a broader range of values. This includes values less than 2, which were not seen as sufficiently useful to standardize. These implementations do not interpret input properly for values of **ibase** that are greater than 16. This is because numeric constants are recognized syntactically, rather than lexically, as described in this volume of IEEE Std 1003.1-200x. They are built from lexical tokens of single hexadecimal digits and periods. Since <blank>s between tokens are not visible at the syntactic level, it is not possible to recognize the multi-digit ‘‘digits’’ used in the higher bases properly. The ability to recognize input in these bases was not considered useful enough to require modifying these implementations. Note that the recognition of numeric constants at the syntactic level is not a problem with conformance to this volume of IEEE Std 1003.1-200x, as it does not impact the behavior of conforming applications (and correct *bc* programs). Historical implementations also accept input with all of the digits '0'–'9' and 'A'–'F' regardless of the value of **ibase**; since digits with value greater than or equal to **ibase** are not really appropriate, the behavior when they appear is undefined, except for the common case of:

```
ibase=8;
    /* Process in octal base. */
...
ibase=A
    /* Restore decimal base. */
```

In some historical implementations, if the expression to be written is an uninitialized array element, a leading <space> and/or up to four leading 0 characters may be output before the character zero. This behavior is considered a bug; it is unlikely that any currently conforming application relies on:

```
echo 'b[3]' | bc
```

returning 00000 rather than 0.

Exact calculation of the number of fractional digits to output for a given value in a base other than 10 can be computationally expensive. Historical implementations use a faster approximation, and this is permitted. Note that the requirements apply only to values of **obase** that this volume of IEEE Std 1003.1-200x requires implementations to support (in particular, not to 1, 0, or negative bases, if an implementation supports them as an extension).

Historical implementations of *bc* did not allow array parameters to be passed as the last parameter to a function. New implementations are encouraged to remove this restriction even though it is not required by the grammar.

FUTURE DIRECTIONS

None.

SEE ALSO

awk

CHANGE HISTORY

First released in Issue 4.

Issue 5

FUTURE DIRECTIONS section added.

Issue 6

Updated to align with the IEEE P1003.2b draft standard, which included resolution of several interpretations of the ISO POSIX-2: 1993 standard.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

8075 **NAME**

8076 bg — run jobs in the background

8077 **SYNOPSIS**

8078 UP `bg [job_id ...]`

8079

8080 **DESCRIPTION**

8081 If job control is enabled (see the description of *set* –**m**), the *bg* utility shall resume suspended jobs
8082 from the current environment (see Section 2.12 (on page 2263)) by running them as background
8083 jobs. If the job specified by *job_id* is already a running background job, the *bg* utility shall have no
8084 effect and shall exit successfully.

8085 Using *bg* to place a job into the background shall cause its process ID to become ‘‘known in the
8086 current shell execution environment’’, as if it had been started as an asynchronous list; see
8087 Section 2.9.3.1 (on page 2252).

8088 **OPTIONS**

8089 None.

8090 **OPERANDS**

8091 The following operand shall be supported:

8092 *job_id* Specify the job to be resumed as a background job. If no *job_id* operand is given,
8093 the most recently suspended job shall be used. The format of *job_id* is described in
8094 the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.203, Job Control Job
8095 ID.

8096 **STDIN**

8097 Not used.

8098 **INPUT FILES**

8099 None.

8100 **ENVIRONMENT VARIABLES**

8101 The following environment variables shall affect the execution of *bg*:

8102 *LANG* Provide a default value for the internationalization variables that are unset or null.
8103 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
8104 Internationalization Variables for the precedence of internationalization variables
8105 used to determine the values of locale categories.)

8106 *LC_ALL* If set to a non-empty string value, override the values of all the other
8107 internationalization variables.

8108 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
8109 characters (for example, single-byte as opposed to multi-byte characters in
8110 arguments).

8111 *LC_MESSAGES*
8112 Determine the locale that should be used to affect the format and contents of
8113 diagnostic messages written to standard error.

8114 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

8115 **ASYNCHRONOUS EVENTS**

8116 Default.

STDOUT

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.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

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.

APPLICATION USAGE

A job is generally suspended by typing the SUSP character (<control>-Z on most systems); see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface. At that point, *bg* can put the job into the background. This is most effective when the job is expecting no terminal input and its output has been redirected to non-terminal files. A background job can be forced to stop when it has terminal output by issuing the command:

```
stty tostop
```

A background job can be stopped with the command:

```
kill –s stop job ID
```

The *bg* utility does not work as expected when it is operating in its own utility execution environment because that environment has no suspended jobs. In the following examples:

```
... | xargs bg
(bg)
```

each *bg* operates in a different environment and does not share its parent shell's understanding of jobs. For this reason, *bg* is generally implemented as a shell regular built-in.

EXAMPLES

None.

RATIONALE

The extensions to the shell specified in this volume of IEEE Std 1003.1-200x 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 standard developers 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 volume of IEEE Std 1003.1-200x to maintain a degree of uniformity with the rest of the KornShell features selected (such as the very popular command line editing features).

The *bg* utility is expected to wrap its output if the output exceeds the number of display columns.

FUTURE DIRECTIONS

None.

SEE ALSO

fg, *kill*, *jobs*, *wait*

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The JC margin marker on the SYNOPSIS is removed since support for Job Control is mandatory in this issue. This is a FIPS requirement.

NAME

c99 — compile standard C programs

SYNOPSIS

CD

```
c99 [–c][–D name[=value]]...[–E][–g][–I directory] ... [–L directory]
    ... [–o outfile][–Ooptlevel][–s][–U name]... operand ...
```

DESCRIPTION

The *c99* utility is an interface to the standard C compilation system; it shall accept source code conforming to the ISO C standard. The system conceptually consists of a compiler and link editor. The files referenced by *operand*s shall be compiled and linked to produce an executable file. (It is unspecified whether the linking occurs entirely within the operation of *c99*; some implementations may produce objects that are not fully resolved until the file is executed.)

If the **–c** option is specified, for all pathname operands of the form *file***.c**, the files:

```
$(basename pathname .c).o
```

shall be created as the result of successful compilation. If the **–c** option is not specified, it is unspecified whether such **.o** files are created or deleted for the *file***.c** operands.

If there are no options that prevent link editing (such as **–c** or **–E**), and all operands compile and link without error, the resulting executable file shall be written according to the **–o** *outfile* option (if present) or to the file **a.out**.

The executable file shall be created as specified in Section 1.7.1.4 (on page 2204), except that the file permission bits shall be set to:

S_IRWXO | S_IRWXG | S_IRWXU

and the bits specified by the *umask* of the process shall be cleared.

OPTIONS

The *c99* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines, except that:

- The **–l** *library* operands have the format of options, but their position within a list of operands affects the order in which libraries are searched.
- The order of specifying the **–I** and **–L** options is significant.
- Conforming applications shall specify each option separately; that is, grouping option letters (for example, **–cO**) need not be recognized by all implementations.

The following options shall be supported:

–c Suppress the link-edit phase of the compilation, and do not remove any object files that are produced.

–g Produce symbolic information in the object or executable files; the nature of this information is unspecified, and may be modified by implementation-defined interactions with other options.

–s Produce object or executable files, or both, from which symbolic and other information not required for proper execution using the *exec* family defined in the System Interfaces volume of IEEE Std 1003.1-200x, has been removed (stripped). If both **–g** and **–s** options are present, the action taken is unspecified.

–o *outfile* Use the pathname *outfile*, instead of the default **a.out**, for the executable file produced. If the **–o** option is present with **–c** or **–E**, the result is unspecified.

8217 –**D** *name*[=*value*]
8218 Define *name* as if by a C-language **#define** directive. If no =*value* is given, a value of
8219 1 shall be used. The –**D** option has lower precedence than the –**U** option. That is, if
8220 *name* is used in both a –**U** and a –**D** option, *name* shall be undefined regardless of
8221 the order of the options. Additional implementation-defined *name*s may be
8222 provided by the compiler. Implementations shall support at least 2 048 bytes of –**D**
8223 definitions and 256 *names*.

8224 –**E** Copy C-language source files to standard output, expanding all preprocessor
8225 directives; no compilation shall be performed. If any operand is not a text file, the
8226 effects are unspecified.

8227 –**I** *directory* Change the algorithm for searching for headers whose names are not absolute
8228 pathnames to look in the directory named by the *directory* pathname before
8229 looking in the usual places. Thus, headers whose names are enclosed in double-
8230 quotes (" ") shall be searched for first in the directory of the file with the **#include**
8231 line, then in directories named in –**I** options, and last in the usual places. For
8232 headers whose names are enclosed in angle brackets ("<>"), the header shall be
8233 searched for only in directories named in –**I** options and then in the usual places.
8234 Directories named in –**I** options shall be searched in the order specified.
8235 Implementations shall support at least ten instances of this option in a single *c99*
8236 command invocation.

8237 –**L** *directory* Change the algorithm of searching for the libraries named in the –**l** objects to look
8238 in the directory named by the *directory* pathname before looking in the usual
8239 places. Directories named in –**L** options shall be searched in the order specified.
8240 Implementations shall support at least ten instances of this option in a single *c99*
8241 command invocation. If a directory specified by a –**L** option contains files named
8242 **libc.a**, **libm.a**, **libl.a**, or **liby.a**, the results are unspecified.

8243 –**O** *optlevel* Specify the level of code optimization. If the *optlevel* option-argument is the digit
8244 `'0'`, all special code optimizations shall be disabled. If it is the digit `'1'`, the
8245 nature of the optimization is unspecified. If the –**O** option is omitted, the nature of
8246 the system's default optimization is unspecified. It is unspecified whether code
8247 generated in the presence of the –**O** 0 option is the same as that generated when
8248 –**O** is omitted. Other *optlevel* values may be supported.

8249 –**U** *name* Remove any initial definition of *name*.

8250 Multiple instances of the –**D**, –**I**, –**U**, and –**L** options can be specified.

8251 **OPERANDS**
8252 An *operand* is either in the form of a pathname or the form –**l** *library*. The application shall
8253 ensure that at least one operand of the pathname form is specified. The following operands shall
8254 be supported:

8255 *file*.**c** A C-language source file to be compiled and optionally linked. The application
8256 shall ensure that the operand is of this form if the –**c** option is used.

8257 *file*.**a** A library of object files typically produced by the *ar* utility, and passed directly to
8258 the link editor. Implementations may recognize implementation-defined suffixes
8259 other than **.a** as denoting object file libraries.

8260 *file*.**o** An object file produced by *c99* –**c** and passed directly to the link editor.
8261 Implementations may recognize implementation-defined suffixes other than **.o** as
8262 denoting object files.

The processing of other files is implementation-defined.

–l library (The letter ell.) Search the library named:

`lib`*`library`*`.a`

A library shall be searched when its name is encountered, so the placement of a **–l** operand is significant. Several standard libraries can be specified in this manner, as described in the EXTENDED DESCRIPTION section. Implementations may recognize implementation-defined suffixes other than **.a** as denoting libraries.

STDIN

Not used.

INPUT FILES

The input file shall be one of the following: a text file containing a C-language source program, an object file in the format produced by *c99* **–c**, or a library of object files, in the format produced by archiving zero or more object files, using *ar*. Implementations may supply additional utilities that produce files in these formats. Additional input file formats are implementation-defined.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *c99*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

TMPDIR Provide a pathname that should override the default directory for temporary files, if any. XSI On XSI-conforming systems, provide a pathname that shall override the default directory for temporary files, if any.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If more than one *file* operand ending in **.c** (or possibly other unspecified suffixes) is given, for each such file:

`"%s:\n", <`*`file`*`>`

may be written. These messages, if written, shall precede the processing of each input file; they shall not be written to the standard output if they are written to the standard error, as described in the STDERR section.

If the **–E** option is specified, the standard output shall be a text file that represents the results of the preprocessing stage of the language; it may contain extra information appropriate for subsequent compilation passes.

STDERR

The standard error shall be used only for diagnostic messages. If more than one *file* operand ending in **.c** (or possibly other unspecified suffixes) is given, for each such file:

```
"%s:\n", <file>
```

may be written to allow identification of the diagnostic and warning messages with the appropriate input file. These messages, if written, shall precede the processing of each input file; they shall not be written to the standard error if they are written to the standard output, as described in the STDOUT section.

This utility may produce warning messages about certain conditions that do not warrant returning an error (non-zero) exit value.

OUTPUT FILES

Object files or executable files or both are produced in unspecified formats.

EXTENDED DESCRIPTION

Standard Libraries

The *c99* utility shall recognize the following **–l** operands for standard libraries:

–l c This operand shall make visible all functions referenced in the System Interfaces volume of IEEE Std 1003.1-200x, with the possible exception of those functions listed as residing in **<aio.h>**, **<arpa/inet.h>**, **<math.h>**, **<mqueue.h>**, **<netdb.h>**, **<netinet/in.h>**, **<pthread.h>**, **<sched.h>**, **<semaphore.h>**, **<spawn.h>**, **<sys/socket.h>**, *pthread_kill*( ), and *pthread_sigmask*( ) in **<signal.h>**, functions marked as extensions other than as part of the MF or MPR extensions in **<sys/mman.h>**, functions marked as ADV in **<fcntl.h>**, and functions marked as CS, CPT, and TMR in **<time.h>**. This operand shall not be required to be present to cause a search of this library.

–l l This operand shall make visible all functions required by the C-language output of *lex* that are not made available through the **–l c** operand.

–l pthread This operand shall make visible all functions referenced in **<pthread.h>** and *pthread_kill*( ) and *pthread_sigmask*( ) referenced in **<signal.h>**. An implementation may search this library in the absence of this operand.

–l m This operand shall make visible all functions referenced in **<math.h>**. An implementation may search this library in the absence of this operand.

–l rt This operand shall make visible all functions referenced in **<aio.h>**, **<mqueue.h>**, **<sched.h>**, **<semaphore.h>**, and **<spawn.h>**, functions marked as extensions other than as part of the MF or MPR extensions in **<sys/mman.h>**, functions marked as ADV in **<fcntl.h>**, and functions marked as CS, CPT, and TMR in **<time.h>**. An implementation may search this library in the absence of this operand.

–l trace This operand shall make visible all functions referenced in **<trace.h>**. An implementation may search this library in the absence of this operand.

–l xnet This operand makes visible all functions referenced in **<arpa/inet.h>**, **<netdb.h>**, **<netinet/in.h>**, and **<sys/socket.h>**. An implementation may search this library in the absence of this operand.

–l y This operand shall make visible all functions required by the C-language output of *yacc* that are not made available through the **–l c** operand.

In the absence of options that inhibit invocation of the link editor, such as **–c** or **–E**, the *c99* utility shall cause the equivalent of a **–l c** operand to be passed to the link editor as the last **–l** operand, causing it to be searched after all other object files and libraries are loaded.

It is unspecified whether the libraries **libc.a**, **libm.a**, **librt.a**, **libpthread.a**, **libl.a**, **liby.a**, or **libxnet** exist as regular files. The implementation may accept as **–l** operands names of objects that do not exist as regular files.

External Symbols

The C compiler and link editor shall support the significance of external symbols up to a length of at least 31 bytes; the action taken upon encountering symbols exceeding the implementation-defined maximum symbol length is unspecified.

The compiler and link editor shall support a minimum of 511 external symbols per source or object file, and a minimum of 4095 external symbols in total. A diagnostic message shall be written to the standard output if the implementation-defined limit is exceeded; other actions are unspecified.

Programming Environments

All implementations shall support one of the following programming environments as a default. Implementations may support more than one of the following programming environments. Applications can use *sysconf*( ) or *getconf* to determine which programming environments are supported.

Table 4-4 Programming Environments: Type Sizes

Programming Environment *getconf* **Name**	**Bits in int**	**Bits in long**	**Bits in pointer**	**Bits in off_t**
_POSIX_V6_ILP32_OFF32	32	32	32	32
_POSIX_V6_ILP32_OFFBIG	32	32	32	≥64
_POSIX_V6_LP64_OFF64	32	64	64	64
_POSIX_V6_LPBIG_OFFBIG	≥32	≥64	≥64	≥64

All implementations shall support one or more environments where the widths of the following types are no greater than the width of type **long**:

blksize_t, **cc_t**, **mode_t**, **nfds_t**, **pid_t**, **ptrdiff_t**, **size_t**, **speed_t**, **ssize_t**, **suseconds_t**, **tcflag_t**, **useconds_t**, **wchar_t**, **wint_t**

The executable files created when these environments are selected shall be in a proper format for execution by the *exec* family of functions. Each environment may be one of the ones in Table 4-4, or it may be another environment. The names for the environments that meet this requirement shall be output by a *getconf* command using the _POSIX_V6_WIDTH_RESTRICTED_ENVS argument. If more than one environment meets the requirement, the names of all such environments shall be output on separate lines. Any of these names can then be used in a subsequent *getconf* command to obtain the flags specific to that environment with the following suffixes added as appropriate:

_CFLAGS To get the C compiler flags.

_LDFLAGS To get the linker/loader flags.

_LIBS To get the libraries.

This requirement may be removed in a future version of IEEE Std 1003.1.

When this utility processes a file containing a function called *main*(), it shall be defined with a return type equivalent to **int**. Using return from *main*() shall be equivalent (other than with respect to language scope issues) to calling *exit*() with the returned value. Reaching the end of *main*() shall be equivalent to calling *exit*(0). The implementation shall not declare a prototype for this function.

Implementations provide configuration strings for C compiler flags, linker/loader flags, and libraries for each supported environment. When an application needs to use a specific programming environment rather than the implementation default programming environment while compiling, the application shall first verify that the implementation supports the desired environment. If the desired programming environment is supported, the application shall then invoke *c99* with the appropriate C compiler flags as the first options for the compile, the appropriate linker/loader flags after any other options but before any operands, and the appropriate libraries at the end of the operands.

Conforming applications shall not attempt to link together object files compiled for different programming models. Applications shall also be aware that binary data placed in shared memory or in files might not be recognized by applications built for other programming models.

Table 4-5 Programming Environments: c99 and cc Arguments

Programming Environment *getconf* **Name**	**Use**	*c99* **and** *cc* **Arguments** *getconf* **Name**
_POSIX_V6_ILP32_OFF32	C Compiler Flags	POSIX_V6_ILP32_OFF32_CFLAGS
	Linker/Loader Flags	POSIX_V6_ILP32_OFF32_LDFLAGS
	Libraries	POSIX_V6_ILP32_OFF32_LIBS
_POSIX_V6_ILP32_OFFBIG	C Compiler Flags	POSIX_V6_ILP32_OFFBIG_CFLAGS
	Linker/Loader Flags	POSIX_V6_ILP32_OFFBIG_LDFLAGS
	Libraries	POSIX_V6_ILP32_OFFBIG_LIBS
_POSIX_V6_LP64_OFF64	C Compiler Flags	POSIX_V6_LP64_OFF64_CFLAGS
	Linker/Loader Flags	POSIX_V6_LP64_OFF64_LDFLAGS
	Libraries	POSIX_V6_LP64_OFF64_LIBS
_POSIX_V6_LPBIG_OFFBIG	C Compiler Flags	POSIX_V6_LPBIG_OFFBIG_CFLAGS
	Linker/Loader Flags	POSIX_V6_LPBIG_OFFBIG_LDFLAGS
	Libraries	POSIX_V6_LPBIG_OFFBIG_LIBS

EXIT STATUS

The following exit values shall be returned:

0 Successful compilation or link edit.

>0 An error occurred.

CONSEQUENCES OF ERRORS

When *c99* encounters a compilation error that causes an object file not to be created, it shall write a diagnostic to standard error and continue to compile other source code operands, but it shall not perform the link phase and return a non-zero exit status. If the link edit is unsuccessful, a diagnostic message shall be written to standard error and *c99* exits with a non-zero status. A conforming application shall rely on the exit status of *c99*, rather than on the existence or mode of the executable file.

APPLICATION USAGE

Since the *c99* utility usually creates files in the current directory during the compilation process, it is typically necessary to run the *c99* utility in a directory in which a file can be created.

On systems providing POSIX Conformance (see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 2, Conformance), *c99* is required only with the C-Language Development option; XSI-conformant systems always provide *c99*.

Some historical implementations have created **.o** files when **–c** is not specified and more than one source file is given. Since this area is left unspecified, the application cannot rely on **.o** files being created, but it also must be prepared for any related **.o** files that already exist being deleted at the completion of the link edit.

Some historical implementations have permitted **–L** options to be interspersed with **–l** operands on the command line. For an application to compile consistently on systems that do not behave like this, it is necessary for a conforming application to supply all **–L** options before any of the **–l** options.

There is the possible implication that if a user supplies versions of the standard functions (before they would be encountered by an implicit **–l c** or explicit **–l m**), that those versions would be used in place of the standard versions. There are various reasons this might not be true (functions defined as macros, manipulations for clean name space, and so on), so the existence of files named in the same manner as the standard libraries within the **–L** directories is explicitly stated to produce unspecified behavior.

All of the functions specified in the System Interfaces volume of IEEE Std 1003.1-200x may be made visible by implementations when the Standard C Library is searched. Conforming applications must explicitly request searching the other standard libraries when functions made visible by those libraries are used.

EXAMPLES

1. The following usage example compiles **foo.c** and creates the executable file **foo**:

   ```
   c99 –o foo foo.c
   ```

   The following usage example compiles **foo.c** and creates the object file **foo.o**:

   ```
   c99 –c foo.c
   ```

   The following usage example compiles **foo.c** and creates the executable file **a.out**:

   ```
   c99 foo.c
   ```

   The following usage example compiles **foo.c**, links it with **bar.o**, and creates the executable file **a.out**. It also creates and leaves **foo.o**:

   ```
   c99 foo.c bar.o
   ```

2. The following example shows how an application using threads interfaces can test for support of and use a programming environment supporting 32-bit **int**, **long**, and **pointer** types and an **off_t** type using at least 64 bits:

   ```
   if [ $(getconf _POSIX_V6_ILP32_OFFBIG) != "–1" ]
   then
       c99 $(getconf POSIX_V6_ILP32_OFFBIG_CFLAGS) –D_XOPEN_SOURCE=600 \
           $(getconf POSIX_V6_ILP32_OFFBIG_LDFLAGS) foo.c –o foo \
           $(getconf POSIX_V6_ILP32_OFFBIG_LIBS) –l pthread
   else
       echo ILP32_OFFBIG programming environment not supported
   ```

```
        exit 1
    fi
```

3. The following examples clarify the use and interactions of **–L** options and **–l** operands.

   Consider the case in which module **a.c** calls function *f*( ) in library **libQ.a**, and module **b.c** calls function *g*( ) in library **libp.a**. Assume that both libraries reside in **/a/b/c**. The command line to compile and link in the desired way is:

   ```
   c99 –L /a/b/c main.o a.c –l Q b.c –l p
   ```

   In this case the **–l Q** operand need only precede the first **–l p** operand, since both **libQ.a** and **libp.a** reside in the same directory.

   Multiple **–L** operands can be used when library name collisions occur. Building on the previous example, suppose that the user wants to use a new **libp.a**, in **/a/a/a**, but still wants *f*( ) from **/a/b/c/libQ.a**:

   ```
   c99 –L /a/a/a –L /a/b/c main.o a.c –l Q b.c –l p
   ```

   In this example, the linker searches the **–L** options in the order specified, and finds **/a/a/a/libp.a** before **/a/b/c/libp.a** when resolving references for **b.c**. The order of the **–l** operands is still important, however.

4. The following example shows how an application can use a programming environment where the widths of the following types:

   blksize_t, **cc_t**, **mode_t**, **nfds_t**, **pid_t**, **ptrdiff_t**, **size_t**, **speed_t**, **ssize_t**, **suseconds_t**, **tcflag_t**, **useconds_t**, **wchar_t**, **wint_t**

   are no greater than the width of type **long**:

```
# First choose one of the listed environments ...
# ... if there are no additional constraints, the first one will do:
CENV=$(getconf _POSIX_V6_WIDTH_RESTRICTED_ENVS | head -n l)
# ... or, if an environment that supports large files is preferred,
# look for names that contain "OFF64" or "OFFBIG". (This chooses
# the last one in the list if none match):
for CENV in $(getconf _POSIX_V6_WIDTH_RESTRICTED_ENVS)
do
    case $CENV in
    *OFF64*|*OFFBIG*) break ;;
    esac
done
# The chosen environment name can now be used like this:
c99 $(getconf ${CENV}_CFLAGS) -D _POSIX_C_SOURCE=200xxxL \
$(getconf ${CENV}_LDFLAGS) foo.c -o foo \
$(getconf ${CENV} _LIBS)
```

RATIONALE

The *c99* utility is based on the *c89* utility originally introduced in the ISO POSIX-2: 1993 standard.

Some of the changes from *c89* include the modification to the contents of the Standard Libraries section to account for new headers and options; for example, **<spawn.h>** added to the **–l rt** operand, and the **–l** trace operand added for the Tracing functions.

FUTURE DIRECTIONS

None.

SEE ALSO

ar, *getconf*, *make*, *nm*, *strip*, *umask*, the System Interfaces volume of IEEE Std 1003.1-200x, *sysconf*( ), the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 13, Headers

CHANGE HISTORY

First released in Issue 6. Included for alignment with the ISO / IEC 9899: 1999 standard.

8527 **NAME**

8528 cal — print a calendar

8529 **SYNOPSIS**

8530 XSI `cal [[`*month*`] `*year*` ]`

8531

8532 **DESCRIPTION**

8533 The *cal* utility shall write a calendar to standard output using the Julian calendar for dates from
8534 January 1, 1 through September 2, 1752 and the Gregorian calendar for dates from September 14,
8535 1752 through December 31, 9999 as though the Gregorian calendar had been adopted on
8536 September 14, 1752.

8537 **OPTIONS**

8538 None.

8539 **OPERANDS**

8540 The following operands shall be supported:

8541 *month* Specify the month to be displayed, represented as a decimal integer from 1
8542 (January) to 12 (December). The default shall be the current month.

8543 *year* Specify the year for which the calendar is displayed, represented as a decimal
8544 integer from 1 to 9999. The default shall be the current year.

8545 **STDIN**

8546 Not used.

8547 **INPUT FILES**

8548 None.

8549 **ENVIRONMENT VARIABLES**

8550 The following environment variables shall affect the execution of *cal*:

8551 *LANG* Provide a default value for the internationalization variables that are unset or null.
8552 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
8553 Internationalization Variables for the precedence of internationalization variables
8554 used to determine the values of locale categories.)

8555 *LC_ALL* If set to a non-empty string value, override the values of all the other
8556 internationalization variables.

8557 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
8558 characters (for example, single-byte as opposed to multi-byte characters in
8559 arguments).

8560 *LC_MESSAGES*
8561 Determine the locale that should be used to affect the format and contents of
8562 diagnostic messages written to standard error, and informative messages written
8563 to standard output.

8564 *LC_TIME* Determine the format and contents of the calendar.

8565 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

8566 *TZ* Determine the timezone used to calculate the value of the current month.

ASYNCHRONOUS EVENTS
Default.

STDOUT
The standard output shall be used to display the calendar, in an unspecified format.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
Note that:

```
cal 83
```

refers to A.D. 83, not 1983.

EXAMPLES
None.

RATIONALE
None.

FUTURE DIRECTIONS
A future version of IEEE Std 1003.1-200x may support locale-specific recognition of the date of adoption of the Gregorian calendar.

SEE ALSO
None.

CHANGE HISTORY
First released in Issue 2.

Issue 6
The DESCRIPTION is updated to allow for traditional behavior for years before the adoption of the Gregorian calendar.

8601 **NAME**

8602 cat — concatenate and print files

8603 **SYNOPSIS**

8604 `cat [-u][`*file ...*`]`

8605 **DESCRIPTION**

8606 The *cat* utility shall read files in sequence and shall write their contents to the standard output in |
8607 the same sequence. |

8608 **OPTIONS**

8609 The *cat* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
8610 Utility Syntax Guidelines.

8611 The following option shall be supported:

8612 **-u** Write bytes from the input file to the standard output without delay as each is
8613 read.

8614 **OPERANDS**

8615 The following operand shall be supported:

8616 *file* A pathname of an input file. If no *file* operands are specified, the standard input |
8617 shall be used. If a *file* is `'-'`, the *cat* utility shall read from the standard input at |
8618 that point in the sequence. The *cat* utility shall not close and reopen standard input
8619 when it is referenced in this way, but shall accept multiple occurrences of `'-'` as a
8620 *file* operand.

8621 **STDIN**

8622 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is `'-'`. |
8623 See the INPUT FILES section.

8624 **INPUT FILES**

8625 The input files can be any file type.

8626 **ENVIRONMENT VARIABLES**

8627 The following environment variables shall affect the execution of *cat*:

8628 *LANG* Provide a default value for the internationalization variables that are unset or null.
8629 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
8630 Internationalization Variables for the precedence of internationalization variables
8631 used to determine the values of locale categories.)

8632 *LC_ALL* If set to a non-empty string value, override the values of all the other
8633 internationalization variables.

8634 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
8635 characters (for example, single-byte as opposed to multi-byte characters in
8636 arguments).

8637 *LC_MESSAGES*
8638 Determine the locale that should be used to affect the format and contents of
8639 diagnostic messages written to standard error.

8640 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

8641 **ASYNCHRONOUS EVENTS**

8642 Default.

STDOUT

The standard output shall contain the sequence of bytes read from the input files. Nothing else shall be written to the standard output.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All input files were output successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The **–u** option has value in prototyping non-blocking reads from FIFOs. The intent is to support the following sequence:

```
mkfifo foo
cat –u foo > /dev/tty13 &
cat –u > foo
```

It is unspecified whether standard output is or is not buffered in the default case. This is sometimes of interest when standard output is associated with a terminal, since buffering may delay the output. The presence of the **–u** option guarantees that unbuffered I/O is available. It is implementation-defined whether the *cat* utility buffers output if the **–u** option is not specified. Traditionally, the **–u** option is implemented using the equivalent of the *setvbuf*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x.

EXAMPLES

The following command:

```
cat myfile
```

writes the contents of the file **myfile** to standard output.

The following command:

```
cat doc1 doc2 > doc.all
```

concatenates the files **doc1** and **doc2** and writes the result to **doc.all**.

Because of the shell language mechanism used to perform output redirection, a command such as this:

```
cat doc doc.end > doc
```

causes the original data in **doc** to be lost.

The command:

```
cat start – middle – end > file
```

when standard input is a terminal, gets two arbitrary pieces of input from the terminal with a single invocation of *cat*. Note, however, that if standard input is a regular file, this would be equivalent to the command:

```
cat start – middle /dev/null end > file
```

because the entire contents of the file would be consumed by *cat* the first time '–' was used as a *file* operand and an end-of-file condition would be detected immediately when '–' was referenced the second time.

RATIONALE

Historical versions of the *cat* utility include the options –**e**, –**t**, and –**v**, which permit the ends of lines, <tab>s, and invisible characters, respectively, to be rendered visible in the output. The standard developers omitted these options because they provide too fine a degree of control over what is made visible, and similar output can be obtained using a command such as:

```
sed –n –e 's/$/$/' –e l pathname
```

The –**s** option was omitted because it corresponds to different functions in BSD and System V-based systems. The BSD –**s** option to squeeze blank lines can be accomplished by the shell script shown in following example:

```
sed –n '
# Write non-empty lines.
/./   {
      p
      d
      }
# Write a single empty line, then look for more empty lines.
/^$/  p
# Get next line, discard the held <newline> (empty line),
# and look for more empty lines.
:Empty
/^$/  {
      N
      s/.//
      b Empty
      }
# Write the non-empty line before going back to search
# for the first in a set of empty lines.
      p
'
```

The System V –**s** option to silence error messages can be accomplished by redirecting the standard error. Note that the BSD documentation for *cat* uses the term ''blank line'' to mean the same as the POSIX ''empty line'': a line consisting only of a <newline>.

The BSD –**n** option was omitted because similar functionality can be obtained from the –**n** option of the *pr* utility.

FUTURE DIRECTIONS

None.

SEE ALSO

more

CHANGE HISTORY

First released in Issue 2.

8730 **NAME**

8731 cd — change the working directory

8732 **SYNOPSIS**

8733 `cd [–L] [–P] [`*directory*`]`

8734 `cd –`

8735 **DESCRIPTION**

8736 The *cd* utility shall change the working directory of the current shell execution environment (see
8737 Section 2.12 (on page 2263)) by executing the following steps in sequence. (In the following
8738 steps, the symbol **curpath** represents an intermediate value used to simplify the description of
8739 the algorithm used by *cd*. There is no requirement that **curpath** be made visible to the
8740 application.)

8741 1. If no *directory* operand is given and the *HOME* environment variable is empty or
8742 undefined, the default behavior is implementation-defined and no further steps shall be
8743 taken.

8744 2. If no *directory* operand is given and the *HOME* environment variable is set to a non-empty
8745 value, the *cd* utility shall behave as if the directory named in the *HOME* environment
8746 variable was specified as the *directory* operand.

8747 3. If the *directory* operand begins with a slash character, set **curpath** to the operand and
8748 proceed to step 7.

8749 4. If the first component of the *directory* operand is dot or dot-dot, proceed to step 6.

8750 5. Starting with the first pathname in the colon-separated pathnames of *CDPATH* (see the
8751 ENVIRONMENT VARIABLES section) if the pathname is non-null, test if the
8752 concatenation of that pathname, a slash character, and the *directory* operand names a
8753 directory. If the pathname is null, test if the concatenation of dot, a slash character, and the
8754 operand names a directory. In either case, if the resulting string names an existing
8755 directory, set **curpath** to that string and proceed to step 7. Otherwise, repeat this step with
8756 the next pathname in *CDPATH* until all pathnames have been tested.

8757 6. Set **curpath** to the string formed by the concatenation of the value of *PWD* a slash
8758 character, and the operand.

8759 7. If the **–P** option is in effect, the *cd* utility shall perform actions equivalent to the *chdir*( )
8760 function, called with **curpath** as the *path* argument. If these actions succeed, the *PWD*
8761 environment variable shall be set to an absolute pathname for the current working
8762 directory and shall not contain filename components that, in the context of pathname
8763 resolution, refer to a file of type symbolic link. If there is insufficient permission on the new
8764 directory, or on any parent of that directory, to determine the current working directory,
8765 the value of the *PWD* environment variable is unspecified. If the actions equivalent to
8766 *chdir*( ) fail for any reason, the *cd* utility shall display an appropriate error message and not
8767 alter the *PWD* environment variable. Whether the actions equivalent to *chdir*( ) succeed or
8768 fail, no further steps shall be taken.

8769 8. The **curpath** value shall then be converted to canonical form as follows, considering each
8770 component from beginning to end, in sequence:

8771 a. Dot components and any slashes that separate them from the next component shall
8772 be deleted.

8773 b. For each dot-dot component, if there is a preceding component and it is neither root
8774 nor dot-dot, the preceding component, all slashes separating the preceding
8775 component from dot-dot, dot-dot, and all slashes separating dot-dot from the

following component shall be deleted.

c. An implementation may further simplify **curpath** by removing any trailing slash characters that are not also leading slashes, replacing multiple non-leading consecutive slashes with a single slash, and replacing three or or more leading slashes with a single slash. If, as a result of this canonicalization, the **curpath** variable is null, no further steps shall be taken.

9. The *cd* utility shall then perform actions equivalent to the *chdir*( ) function called with **curpath** as the *path* argument. If these actions failed for any reason, the *cd* utility shall display an appropriate error message and no further steps shall be taken. The *PWD* environment variable shall be set to **curpath**.

If, during the execution of the above steps, the *PWD* environment variable is changed, the *OLDPWD* environment variable shall also be changed to the value of the old working directory (that is the current working directory immediately prior to the call to *cd*).

OPTIONS

The *cd* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported by the implementation:

–L Handle the operand dot-dot logically; symbolic link components shall not be resolved before dot-dot components are processed (see steps 5. and 6. in the DESCRIPTION).

–P Handle the operand dot-dot physically; symbolic link components shall be resolved before dot-dot components are processed (see step 4. in the DESCRIPTION).

If both **–L** and **–P** options are specified, the last of these options shall be used and all others ignored. If neither **–L** nor **–P** is specified, the operand shall be handled dot-dot logically; see the DESCRIPTION.

OPERANDS

The following operands shall be supported:

directory An absolute or relative pathname of the directory that shall become the new working directory. The interpretation of a relative pathname by *cd* depends on the **–L** option and the *CDPATH* and *PWD* environment variables. If *directory* is an empty string, the results are unspecified.

– When a hyphen is used as the operand, this shall be equivalent to the command: |

```
cd "$OLDPWD" && pwd
```

which changes to the previous working directory and then writes its name.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *cd*:

CDPATH A colon-separated list of pathnames that refer to directories. The *cd* utility shall use this list in its attempt to change the directory, as described in the DESCRIPTION. An empty string in place of a directory pathname represents the current directory.

If *CDPATH* is not set, it shall be treated as if it were an empty string.

HOME The name of the directory, used when no *directory* operand is specified.

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

OLDPWD A pathname of the previous working directory, used by *cd* –.

PWD This variable shall be set as specified in the DESCRIPTION. If an application sets or unsets the value of *PWD*, the behavior of *cd* is unspecified.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If a non-empty directory name from *CDPATH* is used, or if *cd* – is used, an absolute pathname of the new working directory shall be written to the standard output as follows:

```
"%s\n", <new directory>
```

Otherwise, there shall be no output.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 The directory was successfully changed.

>0 An error occurred.

CONSEQUENCES OF ERRORS

The working directory shall remain unchanged.

APPLICATION USAGE

Since *cd* affects the current shell execution environment, it is always provided as a shell regular built-in. If it is called in a subshell or separate utility execution environment, such as one of the following:

```
(cd /tmp)
nohup cd
find . –exec cd {} \;
```

it does not affect the working directory of the caller's environment.

The user must have execute (search) permission in *directory* in order to change to it.

EXAMPLES

None.

RATIONALE

The use of the *CDPATH* was introduced in the System V shell. Its use is analogous to the use of the *PATH* variable in the shell. The BSD C shell used a shell parameter *cdpath* for this purpose.

A common extension when *HOME* is undefined is to get the login directory from the user database for the invoking user. This does not occur on System V implementations.

Some historical shells, such as the KornShell, took special actions when the directory name contained a dot-dot component, selecting the logical parent of the directory, rather than the actual parent directory; that is, it moved up one level toward the `'//'` in the pathname, remembering what the user typed, rather than performing the equivalent of:

```
chdir("..");
```

In such a shell, the following commands would not necessarily produce equivalent output for all directories:

```
cd .. && ls      ls ..
```

This behavior is not permitted by default because it is not consistent with the definition of dot-dot in most historical practice; that is, while this behavior has been optionally available in the KornShell, other shells have historically not supported this functionality. The logical pathname is stored in the *PWD* environment variable when the *cd* utility completes and this value is used to construct the next directory name if *cd* is invoked with the **–L** option.

FUTURE DIRECTIONS

None.

SEE ALSO

pwd, the System Interfaces volume of IEEE Std 1003.1-200x, *chdir*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The *cd* **–**, *PWD*, and *OLDPWD* are added.

The **–L** and **–P** options are added to align with the IEEE P1003.2b draft standard. This also includes the introduction of a new description to include the effect of these options.

8898 **NAME**

8899 cflow — generate a C-language flowgraph (**DEVELOPMENT**)

8900 **SYNOPSIS**

8901 XSI
```
cflow [–r][–d num][–D name[=def]] ... [–i incl][–I dir] ...
    [–U dir] ... file ...
```
8902

8903

8904 **DESCRIPTION**

8905 The *cflow* utility shall analyze a collection of object files or assembler, C-language, *lex* or *yacc*
8906 source files, and attempt to build a graph, written to standard output, charting the external
8907 references.

8908 **OPTIONS**

8909 The *cflow* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
8910 12.2, Utility Syntax Guidelines, except that the order of the –**D**, –**I**, and –**U** options (which are
8911 identical to their interpretation by *c99*) is significant.

8912 The following options shall be supported:

8913 –**d** *num* Indicate the depth at which the flowgraph is cut off. The application shall ensure
8914 that the argument *num* is a decimal integer. By default this is a very large number
8915 (typically greater than 32 000). Attempts to set the cut-off depth to a non-positive |
8916 integer shall be ignored. |

8917 –**i** *incl* Increase the number of included symbols. The *incl* option-argument is one of the
8918 following characters:

8919 *x* Include external and static data symbols. The default shall be to include only
8920 functions in the flowgraph.

8921 _ (Underscore) Include names that begin with an underscore. The default shall
8922 be to exclude these functions (and data if –**i x** is used).

8923 –**r** Reverse the caller:callee relationship, producing an inverted listing showing the |
8924 callers of each function. The listing shall also be sorted in lexicographical order by |
8925 callee. |

8926 **OPERANDS**

8927 The following operand is supported:

8928 *file* The pathname of a file for which a graph is to be generated. Filenames suffixed by |
8929 **.l** shall shall be taken to be *lex* input, **.y** as *yacc* input, **.c** as *c99* input, and **.i** as the |
8930 output of *c99* –**E**. Such files shall be processed as appropriate, determined by their |
8931 suffix. |

8932 Files suffixed in **.s** (conventionally assembler source) may have more limited |
8933 information extracted from them. |

8934 **STDIN**

8935 Not used.

8936 **INPUT FILES**

8937 The input files shall be object files or assembler, C-language, *lex* or *yacc* source files.

8938 **ENVIRONMENT VARIABLES**

8939 The following environment variables shall affect the execution of *cflow*:

8940 *LANG* Provide a default value for the internationalization variables that are unset or null.
8941 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,

Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the ordering of the output when the **–r** option is used.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
The flowgraph written to standard output shall be formatted as follows:

`"%d %s:%s\n", <reference number>, <global>, <definition>`

Each line of output begins with a reference (that is, line) number, followed by indentation of at least one column position per level. This is followed by the name of the global, a colon, and its definition. Normally globals are only functions not defined as an external or beginning with an underscore; see the OPTIONS section for the **–i** inclusion option. For information extracted from C-language source, the definition consists of an abstract type declaration (for example, **char ***) and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the filename and location counter under which the symbol appeared (for example, *text*).

Once a definition of a name has been written, subsequent references to that name contain only the reference number of the line where the definition can be found. For undefined references, only "<>" shall be written.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE

Files produced by *lex* and *yacc* cause the reordering of line number declarations, and this can confuse *cflow*. To obtain proper results, the input of *yacc* or *lex* must be directed to *cflow*.

EXAMPLES

Given the following in **file.c**:

```
int i;
int f();
int g();
int h();
int
main()
{
    f();
    g();
    f();
}
int
f()
{
    i = h();
}
```

The command:

```
cflow –i x file.c
```

produces the output:

```
1 main: int(), <file.c 6>
2    f: int(), <file.c 13>
3        h: <>
4        i: int, <file.c 1>
5    g: <>
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

c99, *lex*, *yacc*

CHANGE HISTORY

First released in Issue 2.

Issue 6

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

NAME

chgrp — change the file group ownership

SYNOPSIS

```
chgrp –hR group file ...
chgrp –R [–H | –L | –P ] group file ...
```

DESCRIPTION

The *chgrp* utility shall set the group ID of the file named by each *file* operand to the group ID specified by the *group* operand.

For each *file* operand, or, if the **–R** option is used, each file encountered while walking the directory trees specified by the *file* operands, the *chgrp* utility shall perform actions equivalent to the *chown*() function defined in the System Interfaces volume of IEEE Std 1003.1-200x, called with the following arguments:

- The *file* operand shall be used as the *path* argument.
- The user ID of the file shall be used as the *owner* argument.
- The specified group ID shall be used as the *group* argument.

Unless *chgrp* is invoked by a process with appropriate privileges, the set-user-ID and set-group-ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and set-group-ID bits of other file types may be cleared.

OPTIONS

The *chgrp* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported by the implementation:

–h If the system supports group IDs for symbolic links, for each *file* operand that names a file of type symbolic link, *chgrp* shall attempt to set the group ID of the symbolic link instead of the file referenced by the symbolic link. If the system does not support group IDs for symbolic links, for each *file* operand that names a file of type symbolic link, *chgrp* shall do nothing more with the current file and shall go on to any remaining files.

–H If the **–R** option is specified and a symbolic link referencing a file of type directory is specified on the command line, *chgrp* shall change the group of the directory referenced by the symbolic link and all files in the file hierarchy below it.

–L If the **–R** option is specified and a symbolic link referencing a file of type directory is specified on the command line or encountered during the traversal of a file hierarchy, *chgrp* shall change the group of the directory referenced by the symbolic link and all files in the file hierarchy below it.

–P If the **–R** option is specified and a symbolic link is specified on the command line or encountered during the traversal of a file hierarchy, *chgrp* shall change the group ID of the symbolic link if the system supports this operation. The *chgrp* utility shall not follow the symbolic link to any other part of the file hierarchy.

–R Recursively change file group IDs. For each *file* operand that names a directory, *chgrp* shall change the group of the directory and all files in the file hierarchy below it. Unless a **–H**, **–L**, or **–P** option is specified, it is unspecified which of these options will be used as the default.

Specifying more than one of the mutually-exclusive options –**H**, –**L**, and –**P** shall not be considered an error. The last option specified shall determine the behavior of the utility.

OPERANDS

The following operands shall be supported:

group A group name from the group database or a numeric group ID. Either specifies a group ID to be given to each file named by one of the *file* operands. If a numeric *group* operand exists in the group database as a group name, the group ID number associated with that group name is used as the group ID.

file A pathname of a file whose group ID is to be modified.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *chgrp*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 The utility executed successfully and all requested changes were made.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Only the owner of a file or the user with appropriate privileges may change the owner or group of a file.

Some implementations restrict the use of *chgrp* to a user with appropriate privileges when the *group* specified is not the effective group ID or one of the supplementary group IDs of the calling process.

EXAMPLES

None.

RATIONALE

The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it can overflow the range of valid exit status values. The standard developers chose to mask these by specifying only 0 and >0 as exit values.

The functionality of *chgrp* is described substantially through references to *chown*( ). In this way, there is no duplication of effort required for describing the interactions of permissions, multiple groups, and so on.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod, *chown*, the System Interfaces volume of IEEE Std 1003.1-200x, *chown*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

New options **–H**, **–L**, and **–P** are added to align with the IEEE P1003.2b draft standard. These options affect the processing of symbolic links.

IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS section to ‘‘Default.’’.

9136 **NAME**

9137 chmod — change the file modes

9138 **SYNOPSIS**

9139 `chmod [–R] mode file ...`

9140 **DESCRIPTION**

9141 The *chmod* utility shall change any or all of the file mode bits of the file named by each *file*
9142 operand in the way specified by the *mode* operand.

9143 It is implementation-defined whether and how the *chmod* utility affects any alternate or
9144 additional file access control mechanism (see the Base Definitions volume of
9145 IEEE Std 1003.1-200x, Section 4.4, File Access Permissions) being used for the specified file.

9146 Only a process whose effective user ID matches the user ID of the file, or a process with the
9147 appropriate privileges, shall be permitted to change the file mode bits of a file.

9148 **OPTIONS**

9149 The *chmod* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
9150 12.2, Utility Syntax Guidelines.

9151 The following option shall be supported:

9152 **–R** Recursively change file mode bits. For each *file* operand that names a directory,
9153 *chmod* shall change the file mode bits of the directory and all files in the file
9154 hierarchy below it.

9155 **OPERANDS**

9156 The following operands shall be supported:

9157 *mode* Represents the change to be made to the file mode bits of each file named by one of
9158 the *file* operands; see the EXTENDED DESCRIPTION section.

9159 *file* A pathname of a file whose file mode bits shall be modified.

9160 **STDIN**

9161 Not used.

9162 **INPUT FILES**

9163 None.

9164 **ENVIRONMENT VARIABLES**

9165 The following environment variables shall affect the execution of *chmod*:

9166 *LANG* Provide a default value for the internationalization variables that are unset or null.
9167 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
9168 Internationalization Variables for the precedence of internationalization variables
9169 used to determine the values of locale categories.)

9170 *LC_ALL* If set to a non-empty string value, override the values of all the other
9171 internationalization variables.

9172 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
9173 characters (for example, single-byte as opposed to multi-byte characters in
9174 arguments).

9175 *LC_MESSAGES*
9176 Determine the locale that should be used to affect the format and contents of
9177 diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

The *mode* operand shall be either a *symbolic_mode* expression or a non-negative octal integer. The *symbolic_mode* form is described by the grammar later in this section.

Each **clause** shall specify an operation to be performed on the current file mode bits of each *file*. The operations shall be performed on each *file* in the order in which the **clause**s are specified.

The **who** symbols **u**, **g**, and **o** shall specify the *user*, *group*, and *other* parts of the file mode bits, respectively. A **who** consisting of the symbol **a** shall be equivalent to **ugo**.

The **perm** symbols **r**, **w**, and **x** represent the *read*, *write*, and *execute*/*search* portions of file mode bits, respectively. The **perm** symbol **s** shall represent the *set-user-ID-on-execution* (when **who** contains or implies **u**) and *set-group-ID-on-execution* (when **who** contains or implies **g**) bits.

The **perm** symbol **X** shall represent the execute/search portion of the file mode bits if the file is a directory or if the current (unmodified) file mode bits have at least one of the execute bits (S_IXUSR, S_IXGRP, or S_IXOTH) set. It shall be ignored if the file is not a directory and none of the execute bits are set in the current file mode bits.

The **permcopy** symbols **u**, **g**, and **o** shall represent the current permissions associated with the user, group, and other parts of the file mode bits, respectively. For the remainder of this section, **perm** refers to the non-terminals **perm** and **permcopy** in the grammar.

If multiple **actionlist**s are grouped with a single **wholist** in the grammar, each **actionlist** shall be applied in the order specified with that **wholist**. The *op* symbols shall represent the operation performed, as follows:

+ If **perm** is not specified, the '+' operation shall not change the file mode bits.

  If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and other permissions, except for those with corresponding bits in the file mode creation mask of the invoking process, shall be set.

  Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.

– If **perm** is not specified, the '–' operation shall not change the file mode bits.

  If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and other permissions, except for those with corresponding bits in the file mode creation mask of the invoking process, shall be cleared.

  Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be cleared.

= Clear the file mode bits specified by the **who** value, or, if no **who** value is specified, all of the file mode bits specified in this volume of IEEE Std 1003.1-200x.

If **perm** is not specified, the '=' operation shall make no further modifications to the file mode bits.

If **who** is not specified, the file mode bits represented by **perm** for the owner, group, and other permissions, except for those with corresponding bits in the file mode creation mask of the invoking process, shall be set.

Otherwise, the file mode bits represented by the specified **who** and **perm** values shall be set.

When using the symbolic mode form on a regular file, it is implementation-defined whether or not:

- Requests to set the set-user-ID-on-execution or set-group-ID-on-execution bit when all execute bits are currently clear and none are being set are ignored.
- Requests to clear all execute bits also clear the set-user-ID-on-execution and set-group-ID-on-execution bits.
- Requests to clear the set-user-ID-on-execution or set-group-ID-on-execution bits when all execute bits are currently clear are ignored. However, if the command *ls* –**l** *file* writes an *s* in the position indicating that the set-user-ID-on-execution or set-group-ID-on-execution is set, the commands *chmod* **u**–**s** *file* or *chmod* **g**–**s** *file*, respectively, shall not be ignored.

When using the symbolic mode form on other file types, it is implementation-defined whether or not requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are honored.

If the **who** symbol **o** is used in conjunction with the **perm** symbol **s** with no other **who** symbols being specified, the set-user-ID-on-execution and set-group-ID-on-execution bits shall not be modified. It shall not be an error to specify the **who** symbol **o** in conjunction with the **perm** symbol **s**.

For an octal integer *mode* operand, the file mode bits shall be set absolutely.

For each bit set in the octal number, the corresponding file permission bit shown in the following table shall be set; all other file permission bits shall be cleared. For regular files, for each bit set in the octal number corresponding to the set-user-ID-on-execution or the set-group-ID-on-execution, bits shown in the following table shall be set; if these bits are not set in the octal number, they are cleared. For other file types, it is implementation-defined whether or not requests to set or clear the set-user-ID-on-execution or set-group-ID-on-execution bits are honored.

Octal	**Mode Bit**	**Octal**	**Mode Bit**	**Octal**	**Mode Bit**	**Octal**	**Mode Bit**
4000	S_ISUID	**0400**	S_IRUSR	**0040**	S_IRGRP	**0004**	S_IROTH
2000	S_ISGID	**0200**	S_IWUSR	**0020**	S_IWGRP	**0002**	S_IWOTH
		0100	S_IXUSR	**0010**	S_IXGRP	**0001**	S_IXOTH

When bits are set in the octal number other than those listed in the table above, the behavior is unspecified.

Grammar for chmod

The grammar and lexical conventions in this section describe the syntax for the *symbolic_mode* operand. The general conventions for this style of grammar are described in Section 1.10 (on page 2220). A valid *symbolic_mode* can be represented as the non-terminal symbol *symbolic_mode* in the grammar. This formal syntax shall take precedence over the preceding text syntax description.

The lexical processing is based entirely on single characters. Implementations need not allow blank characters within the single argument being processed.

```
%start    symbolic_mode
%%

symbolic_mode    : section
                 | symbolic_mode ',' clause
                 ;

clause           : actionlist
                 | wholist actionlist
                 ;

wholist          : who
                 | wholist who
                 ;

who              : 'u' | 'g' | 'o' | 'a'
                 ;

actionlist       : action
                 | actionlist action
                 ;

action           : op
                 | op permlist
                 | op permcopy
                 ;

permcopy         : 'u' | 'g' | 'o'
                 ;

op               : '+' | '-' | '='
                 ;

permlist         : perm
                 | perm permlist
                 ;

perm             : 'r' | 'w' | 'x' | 'X' | 's'
                 ;
```

EXIT STATUS

The following exit values shall be returned:

0 The utility executed successfully and all requested changes were made.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Some implementations of the *chmod* utility change the mode of a directory before the files in the directory when performing a recursive (**–R** option) change; others change the directory mode after the files in the directory. If an application tries to remove read or search permission for a file hierarchy, the removal attempt fails if the directory is changed first; on the other hand, trying to re-enable permissions to a restricted hierarchy fails if directories are changed last. Users should not try to make a hierarchy inaccessible to themselves.

Some implementations of *chmod* never used the process' *umask* when changing modes; systems conformant with this volume of IEEE Std 1003.1-200x do so when **who** is not specified. Note the difference between:

```
chmod a–w file
```

which removes all write permissions, and:

```
chmod –– –w file
```

which removes write permissions that would be allowed if **file** was created with the same *umask*.

Conforming applications should never assume that they know how the set-user-ID and set-group-ID bits on directories are interpreted.

EXAMPLES

Mode	Results
a+=	Equivalent to *a*+,*a*=; clears all file mode bits.
go+–w	Equivalent to *go*+,*go*–*w*; clears group and other write bits.
g=*o*–*w*	Equivalent to *g*=*o*,*g*–*w*; sets group bit to match other bits and then clears group write bit.
g–*r*+*w*	Equivalent to *g*–*r*,*g*+*w*; clears group read bit and sets group write bit.
=*g*	Sets owner bits to match group bits and sets other bits to match group bits.

RATIONALE

The functionality of *chmod* is described substantially through references to concepts defined in the System Interfaces volume of IEEE Std 1003.1-200x. In this way, there is less duplication of effort required for describing the interactions of permissions. However, the behavior of this utility is not described in terms of the *chmod*( ) function from the System Interfaces volume of IEEE Std 1003.1-200x because that specification requires certain side effects upon alternate file access control mechanisms that might not be appropriate, depending on the implementation.

Implementations that support mandatory file and record locking as specified by the 1984 /usr/group standard historically used the combination of set-group-ID bit set and group execute bit clear to indicate mandatory locking. This condition is usually set or cleared with the symbolic mode **perm** symbol **l** instead of the **perm** symbols **s** and **x** so that the mandatory locking mode is not changed without explicit indication that that was what the user intended. Therefore, the details on how the implementation treats these conditions must be defined in the documentation. This volume of IEEE Std 1003.1-200x does not require mandatory locking (nor does the System Interfaces volume of IEEE Std 1003.1-200x), but does allow it as an extension. However, this volume of IEEE Std 1003.1-200x does require that the *ls* and *chmod* utilities work

consistently in this area. If *ls* **–l** *file* indicates that the set-group-ID bit is set, *chmod* **g–s** *file* must clear it (assuming appropriate privileges exist to change modes).

The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it can overflow the range of valid exit status values. This problem is avoided here by specifying only 0 and >0 as exit values.

The System Interfaces volume of IEEE Std 1003.1-200x indicates that implementation-defined restrictions may cause the S_ISUID and S_ISGID bits to be ignored. This volume of IEEE Std 1003.1-200x allows the *chmod* utility to choose to modify these bits before calling *chmod*( ) (or some function providing equivalent capabilities) for non-regular files. Among other things, this allows implementations that use the set-user-ID and set-group-ID bits on directories to enable extended features to handle these extensions in an intelligent manner.

The **X perm** symbol was adopted from BSD-based systems because it provides commonly desired functionality when doing recursive (**–R** option) modifications. Similar functionality is not provided by the *find* utility. Historical BSD versions of *chmod*, however, only supported **X** with *op*+; it has been extended in this volume of IEEE Std 1003.1-200x because it is also useful with *op*=. (It has also been added for *op*– even though it duplicates **x**, in this case, because it is intuitive and easier to explain.)

The grammar was extended with the *permcopy* non-terminal to allow historical-practice forms of symbolic modes like **o**=**u** –**g** (that is, set the ‘‘other’’ permissions to the permissions of ‘‘owner’’ minus the permissions of ‘‘group’’).

FUTURE DIRECTIONS

None.

SEE ALSO

ls, *umask*, the System Interfaces volume of IEEE Std 1003.1-200x, *chmod*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- Octal modes have been kept and made mandatory despite being marked obsolescent in the previous version of this volume of IEEE Std 1003.1-200x.

IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS section to ‘‘Default.’’.

9378 **NAME**

9379 chown — change the file ownership

9380 **SYNOPSIS**

9381 `chown –hR` *owner*`[:`*group*`]` *file* `...`

9382 `chown –R [–H | –L | –P ]` *owner*`[:`*group*`]` *file* `...`

9383 **DESCRIPTION**

9384 The *chown* utility shall set the user ID of the file named by each *file* operand to the user ID
9385 specified by the *owner* operand.

9386 For each *file* operand, or, if the **–R** option is used, each file encountered while walking the
9387 directory trees specified by the *file* operands, the *chown* utility shall perform actions equivalent to
9388 the *chown*() function defined in the System Interfaces volume of IEEE Std 1003.1-200x, called
9389 with the following arguments:

9390 1. The *file* operand shall be used as the *path* argument.

9391 2. The user ID indicated by the *owner* portion of the first operand shall be used as the *owner*
9392 argument.

9393 3. If the *group* portion of the first operand is given, the group ID indicated by it shall be used
9394 as the *group* argument; otherwise, the group ID of the file shall be used as the *group*
9395 argument.

9396 Unless *chown* is invoked by a process with appropriate privileges, the set-user-ID and set-
9397 group-ID bits of a regular file shall be cleared upon successful completion; the set-user-ID and
9398 set-group-ID bits of other file types may be cleared.

9399 **OPTIONS**

9400 The *chown* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
9401 12.2, Utility Syntax Guidelines.

9402 The following options shall be supported by the implementation:

9403 **–h** If the system supports user IDs for symbolic links, for each *file* operand that names
9404 a file of type symbolic link, *chown* shall attempt to set the user ID of the symbolic
9405 link. If the system supports group IDs for symbolic links, and a group ID was
9406 specified, for each *file* operand that names a file of type symbolic link, *chown* shall
9407 attempt to set the group ID of the symbolic link. If the system does not support
9408 user or group IDs for symbolic links, for each *file* operand that names a file of type
9409 symbolic link, *chown* shall do nothing more with the current file and shall go on to
9410 any remaining files.

9411 **–H** If the **–R** option is specified and a symbolic link referencing a file of type directory
9412 is specified on the command line, *chown* shall change the user ID (and group ID, if
9413 specified) of the directory referenced by the symbolic link and all files in the file
9414 hierarchy below it.

9415 **–L** If the **–R** option is specified and a symbolic link referencing a file of type directory
9416 is specified on the command line or encountered during the traversal of a file
9417 hierarchy, *chown* shall change the user ID (and group ID, if specified) of the
9418 directory referenced by the symbolic link and all files in the file hierarchy below it.

9419 **–P** If the **–R** option is specified and a symbolic link is specified on the command line
9420 or encountered during the traversal of a file hierarchy, *chown* shall change the
9421 owner ID (and group ID, if specified) of the symbolic link if the system supports
9422 this operation. The *chown* utility shall not follow the symbolic link to any other

part of the file hierarchy.

–R Recursively change file user and group IDs. For each *file* operand that names a directory, *chown* shall change the user ID (and group ID, if specified) of the directory and all files in the file hierarchy below it. Unless a **–H**, **–L**, or **–P** option is specified, it is unspecified which of these options will be used as the default.

Specifying more than one of the mutually-exclusive options **–H**, **–L**, and **–P** shall not be considered an error. The last option specified shall determine the behavior of the utility.

OPERANDS

The following operands shall be supported:

*owner***[**:*group***]** A user ID and optional group ID to be assigned to *file*. The *owner* portion of this operand shall be a user name from the user database or a numeric user ID. Either specifies a user ID which shall be given to each file named by one of the *file* operands. If a numeric *owner* operand exists in the user database as a user name, the user ID number associated with that user name shall be used as the user ID. Similarly, if the *group* portion of this operand is present, it shall be a group name from the group database or a numeric group ID. Either specifies a group ID which shall be given to each file. If a numeric group operand exists in the group database as a group name, the group ID number associated with that group name shall be used as the group ID.

file A pathname of a file whose user ID is to be modified.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *chown*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 The utility executed successfully and all requested changes were made.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Only the owner of a file or the user with appropriate privileges may change the owner or group of a file.

Some implementations restrict the use of *chown* to a user with appropriate privileges.

EXAMPLES

None.

RATIONALE

The System V and BSD versions use different exit status codes. Some implementations used the exit status as a count of the number of errors that occurred; this practice is unworkable since it can overflow the range of valid exit status values. These are masked by specifying only 0 and >0 as exit values.

The functionality of *chown* is described substantially through references to functions in the System Interfaces volume of IEEE Std 1003.1-200x. In this way, there is no duplication of effort required for describing the interactions of permissions, multiple groups, and so on.

The 4.3 BSD method of specifying both owner and group was included in this volume of IEEE Std 1003.1-200x because:

- There are cases where the desired end condition could not be achieved using the *chgrp* and *chown* (that only changed the user ID) utilities. (If the current owner is not a member of the desired group and the desired owner is not a member of the current group, the *chown*( ) function could fail unless both owner and group are changed at the same time.)
- Even if they could be changed independently, in cases where both are being changed, there is a 100% performance penalty caused by being forced to invoke both utilities.

The BSD syntax *user*[.*group*] was changed to *user*[:*group*] in this volume of IEEE Std 1003.1-200x because the period is a valid character in login names (as specified by the Base Definitions volume of IEEE Std 1003.1-200x, login names consist of characters in the portable filename character set). The colon character was chosen as the replacement for the period character because it would never be allowed as a character in a user name or group name on historical implementations.

The **–R** option is considered by some observers as an undesirable departure from the historical UNIX system tools approach; since a tool, *find*, already exists to recurse over directories, there

seemed to be no good reason to require other tools to have to duplicate that functionality. However, the **–R** option was deemed an important user convenience, is far more efficient than forking a separate process for each element of the directory hierarchy, and is in widespread historical use.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod, *chgrp*, the System Interfaces volume of IEEE Std 1003.1-200x, *chown*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

New options **–h**, **–H**, **–L**, and **–P** are added to align with the IEEE P1003.2b draft standard. These options affect the processing of symbolic links.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

IEEE PASC Interpretation 1003.2 #172 is applied, changing the CONSEQUENCES OF ERRORS section is changed to ‘‘Default.’’.

NAME

cksum — write file checksums and sizes

SYNOPSIS

```
cksum [file ...]
```

DESCRIPTION

The *cksum* utility shall calculate and write to standard output a cyclic redundancy check (CRC) for each input file, and also write to standard output the number of octets in each file. The CRC used is based on the polynomial used for CRC error checking in the ISO/IEC 8802-3:1996 standard (Ethernet).

The encoding for the CRC checksum is defined by the generating polynomial:

$$G(x)=x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$$

Mathematically, the CRC value corresponding to a given file shall be defined by the following procedure:

1. The n bits to be evaluated are considered to be the coefficients of a mod 2 polynomial $M(x)$ of degree $n-1$. These n bits are the bits from the file, with the most significant bit being the most significant bit of the first octet of the file and the last bit being the least significant bit of the last octet, padded with zero bits (if necessary) to achieve an integral number of octets, followed by one or more octets representing the length of the file as a binary value, least significant octet first. The smallest number of octets capable of representing this integer shall be used.
2. $M(x)$ is multiplied by x^{32} (that is, shifted left 32 bits) and divided by $G(x)$ using mod 2 division, producing a remainder $R(x)$ of degree ≤ 31.
3. The coefficients of $R(x)$ are considered to be a 32-bit sequence.
4. The bit sequence is complemented and the result is the CRC.

OPTIONS

None.

OPERANDS

The following operand shall be supported:

file A pathname of a file to be checked. If no *file* operands are specified, the standard input shall be used.

STDIN

The standard input shall be used only if no *file* operands are specified. See the INPUT FILES section.

INPUT FILES

The input files can be any file type.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *cksum*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
For each file processed successfully, the *cksum* utility shall write in the following format:

`"%u %d %s\n", <checksum>, <# of octets>, <pathname>`

If no *file* operand was specified, the pathname and its leading <space> shall be omitted.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 All files were processed successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
The *cksum* utility is typically used to quickly compare a suspect file against a trusted version of the same, such as to ensure that files transmitted over noisy media arrive intact. However, this comparison cannot be considered cryptographically secure. The chances of a damaged file producing the same CRC as the original are small; deliberate deception is difficult, but probably not impossible.

Although input files to *cksum* can be any type, the results need not be what would be expected on character special device files or on file types not described by the System Interfaces volume of IEEE Std 1003.1-200x. Since this volume of IEEE Std 1003.1-200x does not specify the block size used when doing input, checksums of character special files need not process all of the data in those files.

The algorithm is expressed in terms of a bitstream divided into octets. If a file is transmitted between two systems and undergoes any data transformation (such as changing little-endian byte ordering to big-endian), identical CRC values cannot be expected. Implementations performing such transformations may extend *cksum* to handle such situations.

EXAMPLES

None.

RATIONALE

The following C-language program can be used as a model to describe the algorithm. It assumes that a **char** is one octet. It also assumes that the entire file is available for one pass through the function. This was done for simplicity in demonstrating the algorithm, rather than as an implementation model.

```
static unsigned long crctab[] = {
0x00000000,
0x04c11db7, 0x09823b6e, 0x0d4326d9, 0x130476dc, 0x17c56b6b,
0x1a864db2, 0x1e475005, 0x2608edb8, 0x22c9f00f, 0x2f8ad6d6,
0x2b4bcb61, 0x350c9b64, 0x31cd86d3, 0x3c8ea00a, 0x384fbdbd,
0x4c11db70, 0x48d0c6c7, 0x4593e01e, 0x4152fda9, 0x5f15adac,
0x5bd4b01b, 0x569796c2, 0x52568b75, 0x6a1936c8, 0x6ed82b7f,
0x639b0da6, 0x675a1011, 0x791d4014, 0x7ddc5da3, 0x709f7b7a,
0x745e66cd, 0x9823b6e0, 0x9ce2ab57, 0x91a18d8e, 0x95609039,
0x8b27c03c, 0x8fe6dd8b, 0x82a5fb52, 0x8664e6e5, 0xbe2b5b58,
0xbaea46ef, 0xb7a96036, 0xb3687d81, 0xad2f2d84, 0xa9ee3033,
0xa4ad16ea, 0xa06c0b5d, 0xd4326d90, 0xd0f37027, 0xddb056fe,
0xd9714b49, 0xc7361b4c, 0xc3f706fb, 0xceb42022, 0xca753d95,
0xf23a8028, 0xf6fb9d9f, 0xfbb8bb46, 0xff79a6f1, 0xe13ef6f4,
0xe5ffeb43, 0xe8bccd9a, 0xec7dd02d, 0x34867077, 0x30476dc0,
0x3d044b19, 0x39c556ae, 0x278206ab, 0x23431b1c, 0x2e003dc5,
0x2ac12072, 0x128e9dcf, 0x164f8078, 0x1b0ca6a1, 0x1fcdbb16,
0x018aeb13, 0x054bf6a4, 0x0808d07d, 0x0cc9cdca, 0x7897ab07,
0x7c56b6b0, 0x71159069, 0x75d48dde, 0x6b93dddb, 0x6f52c06c,
0x6211e6b5, 0x66d0fb02, 0x5e9f46bf, 0x5a5e5b08, 0x571d7dd1,
0x53dc6066, 0x4d9b3063, 0x495a2dd4, 0x44190b0d, 0x40d816ba,
0xaca5c697, 0xa864db20, 0xa527fdf9, 0xa1e6e04e, 0xbfa1b04b,
0xbb60adfc, 0xb6238b25, 0xb2e29692, 0x8aad2b2f, 0x8e6c3698,
0x832f1041, 0x87ee0df6, 0x99a95df3, 0x9d684044, 0x902b669d,
0x94ea7b2a, 0xe0b41de7, 0xe4750050, 0xe9362689, 0xedf73b3e,
0xf3b06b3b, 0xf771768c, 0xfa325055, 0xfef34de2, 0xc6bcf05f,
0xc27dede8, 0xcf3ecb31, 0xcbffd686, 0xd5b88683, 0xd1799b34,
0xdc3abded, 0xd8fba05a, 0x690ce0ee, 0x6dcdfd59, 0x608edb80,
0x644fc637, 0x7a089632, 0x7ec98b85, 0x738aad5c, 0x774bb0eb,
0x4f040d56, 0x4bc510e1, 0x46863638, 0x42472b8f, 0x5c007b8a,
0x58c1663d, 0x558240e4, 0x51435d53, 0x251d3b9e, 0x21dc2629,
0x2c9f00f0, 0x285e1d47, 0x36194d42, 0x32d850f5, 0x3f9b762c,
0x3b5a6b9b, 0x0315d626, 0x07d4cb91, 0x0a97ed48, 0x0e56f0ff,
0x1011a0fa, 0x14d0bd4d, 0x19939b94, 0x1d528623, 0xf12f560e,
0xf5ee4bb9, 0xf8ad6d60, 0xfc6c70d7, 0xe22b20d2, 0xe6ea3d65,
0xeba91bbc, 0xef68060b, 0xd727bbb6, 0xd3e6a601, 0xdea580d8,
0xda649d6f, 0xc423cd6a, 0xc0e2d0dd, 0xcda1f604, 0xc960ebb3,
0xbd3e8d7e, 0xb9ff90c9, 0xb4bcb610, 0xb07daba7, 0xae3afba2,
0xaafbe615, 0xa7b8c0cc, 0xa379dd7b, 0x9b3660c6, 0x9ff77d71,
0x92b45ba8, 0x9675461f, 0x8832161a, 0x8cf30bad, 0x81b02d74,
0x857130c3, 0x5d8a9099, 0x594b8d2e, 0x5408abf7, 0x50c9b640,
0x4e8ee645, 0x4a4ffbf2, 0x470cdd2b, 0x43cdc09c, 0x7b827d21,
0x7f436096, 0x7200464f, 0x76c15bf8, 0x68860bfd, 0x6c47164a,
0x61043093, 0x65c52d24, 0x119b4be9, 0x155a565e, 0x18197087,
```

```
0x1cd86d30, 0x029f3d35, 0x065e2082, 0x0b1d065b, 0x0fdc1bec,
0x3793a651, 0x3352bbe6, 0x3e119d3f, 0x3ad08088, 0x2497d08d,
0x2056cd3a, 0x2d15ebe3, 0x29d4f654, 0xc5a92679, 0xc1683bce,
0xcc2b1d17, 0xc8ea00a0, 0xd6ad50a5, 0xd26c4d12, 0xdf2f6bcb,
0xdbee767c, 0xe3a1cbc1, 0xe760d676, 0xea23f0af, 0xeee2ed18,
0xf0a5bd1d, 0xf464a0aa, 0xf9278673, 0xfde69bc4, 0x89b8fd09,
0x8d79e0be, 0x803ac667, 0x84fbdbd0, 0x9abc8bd5, 0x9e7d9662,
0x933eb0bb, 0x97ffad0c, 0xafb010b1, 0xab710d06, 0xa6322bdf,
0xa2f33668, 0xbcb4666d, 0xb8757bda, 0xb5365d03, 0xb1f740b4
};

unsigned long memcrc(const unsigned char *b, size_t n)
{
/*  Input arguments:
 *  const char*   b == byte sequence to checksum
 *  size_t        n == length of sequence
 */
    register unsigned   i, c, s = 0;

    for (i = n; i > 0; --i) {
        c = (unsigned)(*b++);
        s = (s << 8) ^ crctab[(s >> 24) ^ c];
    }

    /* Extend with the length of the string. */
    while (n != 0) {
        c = n & 0377;
        n >>= 8;
        s = (s << 8) ^ crctab[(s >> 24) ^ c];
    }

    return ~s;
}
```

The historical practice of writing the number of ‘‘blocks’’ has been changed to writing the number of octets, since the latter is not only more useful, but also since historical implementations have not been consistent in defining what a ‘‘block’’ meant. Octets are used instead of bytes because bytes can differ in size between systems.

The algorithm used was selected to increase the operational robustness of *cksum*. Neither the System V nor BSD *sum* algorithm was selected. Since each of these was different and each was the default behavior on those systems, no realistic compromise was available if either were selected—some set of historical applications would break. Therefore, the name was changed to *cksum*. Although the historical *sum* commands will probably continue to be provided for many years, programs designed for portability across systems should use the new name.

The algorithm selected is based on that used by the ISO/IEC 8802-3:1996 standard (Ethernet) for the frame check sequence field. The algorithm used does not match the technical definition of a *checksum*; the term is used for historical reasons. The length of the file is included in the CRC calculation because this parallels inclusion of a length field by Ethernet in its CRC, but also because it guards against inadvertent collisions between files that begin with different series of zero octets. The chance that two different files produce identical CRCs is much greater when their lengths are not considered. Keeping the length and the checksum of the file itself separate would yield a slightly more robust algorithm, but historical usage has always been that a single number (the checksum as printed) represents the signature of the file. It was decided that

historical usage was the more important consideration.

Early proposals contained modifications to the Ethernet algorithm that involved extracting table values whenever an intermediate result became zero. This was demonstrated to be less robust than the current method and mathematically difficult to describe or justify.

The calculation used is identical to that given in pseudo-code in the referenced Sarwate article. The pseudo-code rendition is:

```
X <– 0; Y <– 0;
for i <– m –1 step –1 until 0 do
    begin
    T <– X(1) ^ A[i];
    X(1) <– X(0); X(0) <– Y(1); Y(1) <– Y(0); Y(0) <– 0;
    comment: f[T] and f'[T] denote the T-th words in the
        table f and f' ;
    X <– X ^ f[T]; Y <– Y ^ f'[T];
    end
```

The pseudo-code is reproduced exactly as given; however, note that in the case of *cksum*, **A[i]** represents a byte of the file, the words **X** and **Y** are treated as a single 32-bit value, and the tables **f** and **f'** are a single table containing 32-bit values.

The referenced Sarwate article also discusses generating the table.

FUTURE DIRECTIONS

None.

SEE ALSO

None.

CHANGE HISTORY

First released in Issue 4.

NAME

cmp — compare two files

SYNOPSIS

```
cmp [ –l | –s ] file1 file2
```

DESCRIPTION

The *cmp* utility shall compare two files. The *cmp* utility shall write no output if the files are the same. Under default options, if they differ, it shall write to standard output the byte and line number at which the first difference occurred. Bytes and lines shall be numbered beginning with 1.

OPTIONS

The *cmp* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–l (Lowercase ell.) Write the byte number (decimal) and the differing bytes (octal) for each difference.

–s Write nothing for differing files; return exit status only.

OPERANDS

The following operands shall be supported:

file1 A pathname of the first file to be compared. If *file1* is `'–'`, the standard input shall be used.

file2 A pathname of the second file to be compared. If *file2* is `'–'`, the standard input shall be used.

If both *file1* and *file2* refer to standard input or refer to the same FIFO special, block special, or character special file, the results are undefined.

STDIN

The standard input shall be used only if the *file1* or *file2* operand refers to standard input. See the INPUT FILES section.

INPUT FILES

The input files can be any file type.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *cmp*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

In the POSIX locale, results of the comparison shall be written to standard output. When no options are used, the format shall be:

```
"%s %s differ: char %d, line %d\n", file1, file2,
    <byte number>, <line number>
```

When the **–l** option is used, the format shall be:

```
"%d %o %o\n", <byte number>, <differing byte>,
    <differing byte>
```

for each byte that differs. The first <*differing byte*> number is from *file1* while the second is from *file2*. In both cases, <*byte number*> shall be relative to the beginning of the file, beginning with 1.

No output shall be written to standard output when the **–s** option is used.

STDERR

The standard error shall be used only for diagnostic messages. If *file1* and *file2* are identical for the entire length of the shorter file, in the POSIX locale the following diagnostic message shall be written, unless the **–s** option is specified:

```
"cmp: EOF on %s%s\n", <name of shorter file>, <additional info>
```

The <*additional info*> field shall either be null or a string that starts with a <blank> and contains no <newline>s. Some implementations report on the number of lines in this case.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 The files are identical.

1 The files are different; this includes the case where one file is identical to the first part of the other.

>1 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Although input files to *cmp* can be any type, the results might not be what would be expected on character special device files or on file types not described by the System Interfaces volume of IEEE Std 1003.1-200x. Since this volume of IEEE Std 1003.1-200x does not specify the block size used when doing input, comparisons of character special files need not compare all of the data in those files.

For files which are not text files, line numbers simply reflect the presence of a <newline>, without any implication that the file is organized into lines.

EXAMPLES

None.

RATIONALE

The global language in Section 1.11 (on page 2221) indicates that using two mutually-exclusive options together produces unspecified results. Some System V implementations consider the option usage:

```
cmp -l -s ...
```

to be an error. They also treat:

```
cmp -s -l ...
```

as if no options were specified. Both of these behaviors are considered bugs, but are allowed.

The word **char** in the standard output format comes from historical usage, even though it is actually a byte number. When *cmp* is supported in other locales, implementations are encouraged to use the word *byte* or its equivalent in another language. Users should not interpret this difference to indicate that the functionality of the utility changed between locales.

Some implementations report on the number of lines in the identical-but-shorter file case. This is allowed by the inclusion of the <*additional info*> fields in the output format. The restriction on having a leading <blank> and no <newline>s is to make parsing for the filename easier. It is recognized that some filenames containing white-space characters make parsing difficult anyway, but the restriction does aid programs used on systems where the names are predominantly well behaved.

FUTURE DIRECTIONS

None.

SEE ALSO

comm, *diff*

CHANGE HISTORY

First released in Issue 2.

9842 **NAME**

9843 comm — select or reject lines common to two files

9844 **SYNOPSIS**

9845 `comm [–123] file1 file2`

9846 **DESCRIPTION**

9847 The *comm* utility shall read *file1* and *file2*, which should be ordered in the current collating
9848 sequence, and produce three text columns as output: lines only in *file1*, lines only in *file2*, and
9849 lines in both files.

9850 If the lines in both files are not ordered according to the collating sequence of the current locale,
9851 the results are unspecified.

9852 **OPTIONS**

9853 The *comm* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
9854 12.2, Utility Syntax Guidelines.

9855 The following options shall be supported:

9856 **–1** Suppress the output column of lines unique to *file1*.

9857 **–2** Suppress the output column of lines unique to *file2*.

9858 **–3** Suppress the output column of lines duplicated in *file1* and *file2*.

9859 **OPERANDS**

9860 The following operands shall be supported:

9861 *file1* A pathname of the first file to be compared. If *file1* is `'–'`, the standard input shall |
9862 be used. |

9863 *file2* A pathname of the second file to be compared. If *file2* is `'–'`, the standard input |
9864 shall be used. |

9865 If both *file1* and *file2* refer to standard input or to the same FIFO special, block special, or
9866 character special file, the results are undefined.

9867 **STDIN**

9868 The standard input shall be used only if one of the *file1* or *file2* operands refers to standard input.
9869 See the INPUT FILES section.

9870 **INPUT FILES**

9871 The input files shall be text files.

9872 **ENVIRONMENT VARIABLES**

9873 The following environment variables shall affect the execution of *comm*:

9874 *LANG* Provide a default value for the internationalization variables that are unset or null.
9875 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
9876 Internationalization Variables for the precedence of internationalization variables
9877 used to determine the values of locale categories.)

9878 *LC_ALL* If set to a non-empty string value, override the values of all the other
9879 internationalization variables.

9880 *LC_COLLATE*
9881 Determine the locale for the collating sequence *comm* expects to have been used
9882 when the input files were sorted.

9883 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
9884 characters (for example, single-byte as opposed to multi-byte characters in

arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
The *comm* utility shall produce output depending on the options selected. If the **–1**, **–2**, and **–3** options are all selected, *comm* shall write nothing to standard output.

If the **–1** option is not selected, lines contained only in *file1* shall be written using the format:

`"%s\n", <line in file1>`

If the **–2** option is not selected, lines contained only in *file2* are written using the format:

`"%s%s\n", <lead>, <line in file2>`

where the string <*lead*> is as follows:

<tab> The **–1** option is not selected.

null string The **–1** option is selected.

If the **–3** option is not selected, lines contained in both files shall be written using the format:

`"%s%s\n", <lead>, <line in both>`

where the string <*lead*> is as follows:

<tab><tab> Neither the **–1** nor the **–2** option is selected.

<tab> Exactly one of the **–1** and **–2** options is selected.

null string Both the **–1** and **–2** options are selected.

If the input files were ordered according to the collating sequence of the current locale, the lines written shall be in the collating sequence of the original lines.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 All input files were successfully output as specified.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE

If the input files are not properly presorted, the output of *comm* might not be useful.

EXAMPLES

If a file named **xcu** contains a sorted list of the utilities in this volume of IEEE Std 1003.1-200x, a file named **xpg3** contains a sorted list of the utilities specified in the X/Open Portability Guide, Issue 3, and a file named **svid89** contains a sorted list of the utilities in the System V Interface Definition Third Edition:

```
comm –23 xcu xpg3 | comm –23 – svid89
```

would print a list of utilities in this volume of IEEE Std 1003.1-200x not specified by either of the other documents:

```
comm –12 xcu xpg3 | comm –12 – svid89
```

would print a list of utilities specified by all three documents, and:

```
comm –12 xpg3 svid89 | comm –23 – xcu
```

would print a list of utilities specified by both XPG3 and the SVID, but not specified in this volume of IEEE Std 1003.1-200x.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cmp, *diff*, *sort*, *uniq*

CHANGE HISTORY

First released in Issue 2.

Issue 6

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

NAME

command — execute a simple command

SYNOPSIS

```
command [–p] command_name [argument ...]
command [ –v | –V ] command_name
```

DESCRIPTION

The *command* utility shall cause the shell to treat the arguments as a simple command, suppressing the shell function lookup that is described in Section 2.9.1.1 (on page 2249), item 1b.

If the *command_name* is the same as the name of one of the special built-in utilities, the special properties in the enumerated list at the beginning of Section 2.14 (on page 2266) shall not occur. In every other respect, if *command_name* is not the name of a function, the effect of *command* (with no options) shall be the same as omitting *command*.

On systems supporting the User Portability Utilities option, the *command* utility also shall provide information concerning how a command name is interpreted by the shell; see **–v** and **–V**.

OPTIONS

The *command* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–p Perform the command search using a default value for *PATH* that is guaranteed to find all of the standard utilities.

–v (On systems supporting the User Portability Utilities option.) 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 Section 2.12 (on page 2263)), to invoke *command_name*, but do not invoke *command_name*.

- Utilities, regular built-in utilities, *command_name*s including a slash character, and any implementation-defined functions that are found using the *PATH* variable (as described in Section 2.9.1.1 (on page 2249)), 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 (On systems supporting the User Portability Utilities option.) 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 Section 2.12 (on page 2263)), but do not invoke *command_name*. Although the format of this string is unspecified, it shall indicate in which of the following categories *command_name* falls and shall include the information stated:

- Utilities, regular built-in utilities, and any implementation-defined functions that are found using the *PATH* variable (as described in Section 2.9.1.1 (on page 2249)), 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 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* search shall be identified as regular built-in utilities. (The term ‘‘regular’’ need not be used.)
- Shell reserved words shall be identified as reserved words.

OPERANDS

The following operands shall be supported:

argument One of the strings treated as an argument to *command_name*.

command_name
The name of a utility or a special built-in utility.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *command*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

PATH Determine the search path used during the command search described in Section 2.9.1.1 (on page 2249), except as described under the **–p** option.

ASYNCHRONOUS EVENTS

Default.

STDOUT

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>
```

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

When the **–v** or **–V** options are specified, the following exit values shall be returned:

0 Successful completion.

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

Otherwise, the following exit values shall be returned:

126 The utility specified by *command_name* was found but could not be invoked.

127 An error occurred in the *command* utility or the utility specified by *command_name* could not be found.

Otherwise, the exit status of *command* shall be that of the simple command specified by the arguments to *command*.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The order for command search allows functions to override regular built-ins and path searches. This utility is necessary to allow functions that have the same name as a utility to call the utility (instead of a recursive call to the function).

The system default path is available using *getconf*; however, since *getconf* may need to have the *PATH* set up before it can be called itself, the following can be used:

```
command –p getconf _CS_PATH
```

There are some advantages to suppressing the special characteristics of special built-ins on occasion. For example:

```
command exec > unwritable-file
```

does not cause a non-interactive script to abort, so that the output status can be checked by the script.

The *command*, *env*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if an error occurs so that applications can distinguish ‘‘failure to find a utility’’ from ‘‘invoked utility exited with an error indication’’. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for ‘‘normal error conditions’’ and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for any other reason.

Since the **–v** and **–V** options of *command* produce output in relation to the current shell execution environment, *command* is generally provided as a shell regular built-in. If it is called in a subshell

or separate utility execution environment, such as one of the following:

```
(PATH=foo command –v)
 nohup command –v
```

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

Two types of regular built-ins could be encountered on a system and these are described separately by *command*. The description of command search in Section 2.9.1.1 (on page 2249) allows for a standard utility to be implemented as a regular built-in as long as it is found in the appropriate place in a *PATH* search. So, for example, *command* –**v** *true* might yield **/bin/true** or some similar pathname. Other implementation-defined utilities that are not defined by this volume of IEEE Std 1003.1-200x might exist only as built-ins and have no pathname associated with them. These produce output identified as (regular) built-ins. Applications encountering these are not able to count on *exec*ing them, using them with *nohup*, overriding them with a different *PATH*, and so on.

EXAMPLES

1. Make a version of *cd* that always prints out the new working directory exactly once:

```
cd() {
    command cd "$@" >/dev/null
    pwd
}
```

2. Start off a ‘‘secure shell script’’ in which the script avoids being spoofed by its parent:

```
IFS='
'
#    The preceding value should be <space><tab><newline>.
#    Set IFS to its default value.

\unalias –a
#    Unset all possible aliases.
#    Note that unalias is escaped to prevent an alias
#    being used for unalias.

unset –f command
#    Ensure command is not a user function.

PATH="$(command –p getconf _CS_PATH):$PATH"
#    Put on a reliable PATH prefix.

#    ...
```

At this point, given correct permissions on the directories called by *PATH*, the script has the ability to ensure that any utility it calls is the intended one. It is being very cautious because it assumes that implementation extensions may be present that would allow user functions to exist when it is invoked; this capability is not specified by this volume of IEEE Std 1003.1-200x, but it is not prohibited as an extension. For example, the *ENV* variable precedes the invocation of the script with a user start-up script. Such a script could define functions to spoof the application.

RATIONALE

Since *command* is a regular built-in utility it is always found prior to the *PATH* search.

There is nothing in the description of *command* that implies the command line is parsed any differently from that of any other simple command. For example:

```
command a | b ; c
```

is not parsed in any special way that causes `'|'` or `';'` to be treated other than a pipe operator or semicolon or that prevents function lookup on **b** or **c**.

The *command* utility is somewhat similar to the Eighth Edition shell *builtin* command, but since *command* also goes to the file system to search for utilities, the name *builtin* would not be intuitive.

The *command* utility is most likely to be provided as a regular built-in. It is not listed as a special built-in for the following reasons:

- The removal of exportable functions made the special precedence of a special built-in unnecessary.
- A special built-in has special properties (see Section 2.14 (on page 2266)) that were inappropriate for invoking other utilities. For example, two commands such as:

  ```
  date > unwritable-file
  ```

  ```
  command date > unwritable-file
  ```

  would have entirely different results; in a non-interactive script, the former would continue to execute the next command, the latter would abort. Introducing this semantic difference along with suppressing functions was seen to be non-intuitive.

The **–p** option is present because it is useful to be able to ensure a safe path search that finds all the POSIX Shell and Utilities standard utilities. This search might not be identical to the one that occurs through one of the POSIX System Interfaces *exec* functions when *PATH* is unset. At the very least, this feature is required to allow the script to access the correct version of *getconf* so that the value of the default path can be accurately retrieved.

The *command* **–v** and **–V** options were added 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.

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.

FUTURE DIRECTIONS

None.

SEE ALSO

sh, *type*

10159 **CHANGE HISTORY**
10160 First released in Issue 4.

10161 **NAME**

10162 compress — compress data

10163 **SYNOPSIS**

10164 XSI `compress [–fv][–b bits][file ...]`

10165 `compress [–cfv][–b bits][file]`

10166

10167 **DESCRIPTION**

10168 The *compress* utility shall attempt to reduce the size of the named files by using adaptive
10169 Lempel-Ziv coding algorithm.

10170 **Note:** Lempel-Ziv is US Patent 4464650, issued to William Eastman, Abraham Lempel, Jacob Ziv,
10171 Martin Cohn on August 7th, 1984, and assigned to Sperry Corporation. |

10172 Lempel-Ziv-Welch compression is covered by US Patent 4558302, issued to Terry A. Welch on |
10173 December 10th, 1985, and assigned to Sperry Corporation. |

10174 On systems not supporting adaptive Lempel-Ziv coding algorithm, the input files shall not be
10175 changed and an error value greater than two shall be returned. Except when the output is to the
10176 standard output, each file shall be replaced by one with the extension **.Z**. If the invoking process
10177 has appropriate privileges, the ownership, modes, access time, and modification time of the
10178 original file are preserved. If appending the **.Z** to the filename would make the name exceed
10179 {NAME_MAX} bytes, the command shall fail. If no files are specified, the standard input shall be
10180 compressed to the standard output.

10181 **OPTIONS**

10182 The *compress* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x,
10183 Section 12.2, Utility Syntax Guidelines.

10184 The following options shall be supported:

10185 **–b** *bits* Specify the maximum number of bits to use in a code. For a conforming |
10186 application, the *bits* argument shall be:

10187 `9 <= bits <= 14`

10188 The implementation may allow *bits* values of greater than 14. The default is 14, 15,
10189 or 16.

10190 **–c** Cause *compress* to write to the standard output; the input file is not changed, and
10191 no **.Z** files are created.

10192 **–f** Force compression of *file*, even if it does not actually reduce the size of the file, or if
10193 the corresponding *file***.Z** file already exists. If the **–f** option is not given, and the
10194 process is not running in the background, the user is prompted as to whether an
10195 existing *file***.Z** file should be overwritten.

10196 **–v** Write the percentage reduction of each file to standard error.

10197 **OPERANDS**

10198 The following operand shall be supported:

10199 *file* A pathname of a file to be compressed.

10200 **STDIN**

10201 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is `'–'`.

10202 **INPUT FILES**

10203 If *file* operands are specified, the input files contain the data to be compressed.

10204 **ENVIRONMENT VARIABLES**

10205 The following environment variables shall affect the execution of *compress*:

10206 *LANG* Provide a default value for the internationalization variables that are unset or null.
10207 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
10208 Internationalization Variables for the precedence of internationalization variables
10209 used to determine the values of locale categories.)

10210 *LC_ALL* If set to a non-empty string value, override the values of all the other
10211 internationalization variables.

10212 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
10213 characters (for example, single-byte as opposed to multi-byte characters in
10214 arguments).

10215 *LC_MESSAGES*
10216 Determine the locale that should be used to affect the format and contents of
10217 diagnostic messages written to standard error.

10218 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

10219 **ASYNCHRONOUS EVENTS**

10220 Default.

10221 **STDOUT**

10222 If no *file* operands are specified, or if a *file* operand is '**−**', or if the **−c** option is specified, the
10223 standard output contains the compressed output.

10224 **STDERR**

10225 The standard error shall be used only for diagnostic and prompt messages and the output from |
10226 **−v**.

10227 **OUTPUT FILES**

10228 The output files shall contain the compressed output. The format of compressed files is
10229 unspecified and interchange of such files between implementations (including access via
10230 unspecified file sharing mechanisms) is not required by IEEE Std 1003.1-200x.

10231 **EXTENDED DESCRIPTION**

10232 None.

10233 **EXIT STATUS**

10234 The following exit values shall be returned:

10235 0 Successful completion.

10236 1 An error occurred.

10237 2 One or more files were not compressed because they would have increased in size (and the
10238 **−f** option was not specified).

10239 >2 An error occurred.

10240 **CONSEQUENCES OF ERRORS**

10241 The input file shall remain unmodified.

APPLICATION USAGE

The amount of compression obtained depends on the size of the input, the number of *bits* per code, and the distribution of common substrings. Typically, text such as source code or English is reduced by 50-60%. Compression is generally much better than that achieved by Huffman coding or adaptive Huffman coding (*compact*), and takes less time to compute.

Although *compress* strictly follows the default actions upon receipt of a signal or when an error occurs, some unexpected results may occur. In some implementations it is likely that a partially compressed file is left in place, alongside its uncompressed input file. Since the general operation of *compress* is to delete the uncompressed file only after the **.Z** file has been successfully filled, an application should always carefully check the exit status of *compress* before arbitrarily deleting files that have like-named neighbors with **.Z** suffixes.

The limit of 14 on the *bits* option-argument is to achieve portability to all systems (within the restrictions imposed by the lack of an explicit published file format). Some implementations based on 16-bit architectures cannot support 15 or 16-bit uncompression.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

uncompress, *zcat*

CHANGE HISTORY

First released in Issue 4.

Issue 6

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

An error case is added for systems not supporting adaptive Lempel-Ziv coding.

10269 **NAME**

10270 cp — copy files

10271 **SYNOPSIS**

10272 `cp [–fip] source_file target_file`

10273 `cp [–fip] source_file ... target`

10274 `cp –R [–H | –L | –P][–fip] source_file ... target`

10275 OB `cp –r [–H | –L | –P][–fip] source_file ... target`

10276 **DESCRIPTION**

10277 The first synopsis form is denoted by two operands, neither of which are existing files of type
10278 directory. The *cp* utility shall copy the contents of *source_file* (or, if *source_file* is a file of type
10279 symbolic link, the contents of the file referenced by *source_file*) to the destination path named by
10280 *target_file*.

10281 The second synopsis form is denoted by two or more operands where the **–R** or **–r** options are
10282 not specified and the first synopsis form is not applicable. It shall be an error if any *source_file* is a
10283 file of type directory, if *target* does not exist, or if *target* is a file of a type defined by the System
10284 Interfaces volume of IEEE Std 1003.1-200x, but is not a file of type directory. The *cp* utility shall
10285 copy the contents of each *source_file* (or, if *source_file* is a file of type symbolic link, the contents
10286 of the file referenced by *source_file*) to the destination path named by the concatenation of *target*,
10287 a slash character, and the last component of *source_file*.

10288 The third and fourth synopsis forms are denoted by two or more operands where the **–R** or **–r**
10289 options are specified. The *cp* utility shall copy each file in the file hierarchy rooted in each
10290 *source_file* to a destination path named as follows: |

10291 • If *target* exists and is a file of type directory, the name of the corresponding destination path |
10292 for each file in the file hierarchy shall be the concatenation of *target*, a slash character, and the
10293 pathname of the file relative to the directory containing *source_file*. |

10294 • If *target* does not exist and two operands are specified, the name of the corresponding |
10295 destination path for *source_file* shall be *target*; the name of the corresponding destination path
10296 for all other files in the file hierarchy shall be the concatenation of *target*, a slash character,
10297 and the pathname of the file relative to *source_file*. |

10298 It shall be an error if *target* does not exist and more than two operands are specified, or if *target* |
10299 exists and is a file of a type defined by the System Interfaces volume of IEEE Std 1003.1-200x, but
10300 is not a file of type directory.

10301 In the following description, the term *dest_file* refers to the file named by the destination path.
10302 The term *source_file* refers to the file that is being copied, whether specified as an operand or a
10303 file in a file hierarchy rooted in a *source_file* operand. If *source_file* is a file of type symbolic link:

10304 • If neither the **–R** nor **–r** options were specified, *cp* shall take actions based on the type and
10305 contents of the file referenced by the symbolic link, and not by the symbolic link itself.

10306 • If the **–R** option was specified:

10307 — If none of the options **–H**, **–L**, nor **–P** were specified, it is unspecified which of **–H**, **–L**, or
10308 **–P** will be used as a default.

10309 — If the **–H** option was specified, *cp* shall take actions based on the type and contents of the
10310 file referenced by any symbolic link specified as a *source_file* operand.

10311 — If the **–L** option was specified, *cp* shall take actions based on the type and contents of the
10312 file referenced by any symbolic link specified as a *source_file* operand or any symbolic

links encountered during traversal of a file hierarchy.

— If the **–P** option was specified, *cp* shall copy any symbolic link specified as a *source_file* operand and any symbolic links encountered during traversal of a file hierarchy, and shall not follow any symbolic links.

- If the **–r** option was specified, the behavior is implementation-defined.

For each *source_file*, the following steps shall be taken:

1. If *source_file* references the same file as *dest_file*, *cp* may write a diagnostic message to standard error; it shall do nothing more with *source_file* and shall go on to any remaining files.

2. If *source_file* is of type directory, the following steps shall be taken:

   a. If neither the **–R** or **–r** options were specified, *cp* shall write a diagnostic message to standard error, do nothing more with *source_file*, and go on to any remaining files.

   b. If *source_file* was not specified as an operand and *source_file* is dot or dot-dot, *cp* shall do nothing more with *source_file* and go on to any remaining files.

   c. If *dest_file* exists and it is a file type not specified by the System Interfaces volume of IEEE Std 1003.1-200x, the behavior is implementation-defined.

   d. If *dest_file* exists and it is not of type directory, *cp* shall write a diagnostic message to standard error, do nothing more with *source_file* or any files below *source_file* in the file hierarchy, and go on to any remaining files.

   e. If the directory *dest_file* does not exist, it shall be created with file permission bits set to the same value as those of *source_file*, modified by the file creation mask of the user if the **–p** option was not specified, and then bitwise-inclusively OR'ed with S_IRWXU. If *dest_file* cannot be created, *cp* shall write a diagnostic message to standard error, do nothing more with *source_file*, and go on to any remaining files. It is unspecified if *cp* attempts to copy files in the file hierarchy rooted in *source_file*.

   f. The files in the directory *source_file* shall be copied to the directory *dest_file*, taking the four steps (1 to 4) listed here with the files as *source_file*s.

   g. If *dest_file* was created, its file permission bits shall be changed (if necessary) to be the same as those of *source_file*, modified by the file creation mask of the user if the **–p** option was not specified.

   h. The *cp* utility shall do nothing more with *source_file* and go on to any remaining files.

3. If *source_file* is of type regular file, the following steps shall be taken:

   a. If *dest_file* exists, the following steps shall be taken:

      i. If the **–i** option is in effect, the *cp* utility shall write a prompt to the standard error and read a line from the standard input. If the response is not affirmative, *cp* shall do nothing more with *source_file* and go on to any remaining files.

      ii. A file descriptor for *dest_file* shall be obtained by performing actions equivalent to the *open*() function defined in the System Interfaces volume of IEEE Std 1003.1-200x called using *dest_file* as the *path* argument, and the bitwise-inclusive OR of O_WRONLY and O_TRUNC as the *oflag* argument.

      iii. If the attempt to obtain a file descriptor fails and the **–f** option is in effect, *cp* shall attempt to remove the file by performing actions equivalent to the *unlink*() function defined in the System Interfaces volume of

IEEE Std 1003.1-200x called using *dest_file* as the *path* argument. If this attempt succeeds, *cp* shall continue with step 3b.

b. If *dest_file* does not exist, a file descriptor shall be obtained by performing actions equivalent to the *open*() function defined in the System Interfaces volume of IEEE Std 1003.1-200x called using *dest_file* as the *path* argument, and the bitwise-inclusive OR of O_WRONLY and O_CREAT as the *oflag* argument. The file permission bits of *source_file* shall be the *mode* argument.

c. If the attempt to obtain a file descriptor fails, *cp* shall write a diagnostic message to standard error, do nothing more with *source_file*, and go on to any remaining files.

d. The contents of *source_file* shall be written to the file descriptor. Any write errors shall cause *cp* to write a diagnostic message to standard error and continue to step 3e.

e. The file descriptor shall be closed.

f. The *cp* utility shall do nothing more with *source_file*. If a write error occurred in step 3d, it is unspecified if *cp* continues with any remaining files. If no write error occurred in step 3d, *cp* shall go on to any remaining files.

4. Otherwise, the following steps shall be taken:

a. If the **–r** option was specified, the behavior is implementation-defined.

b. If the **–R** option was specified, the following steps shall be taken:

i. The *dest_file* shall be created with the same file type as *source_file*.

ii. If *source_file* is a file of type FIFO, the file permission bits shall be the same as those of *source_file*, modified by the file creation mask of the user if the **–p** option was not specified. Otherwise, the permissions, owner ID, and group ID of *dest_file* are implementation-defined.

If this creation fails for any reason, *cp* shall write a diagnostic message to standard error, do nothing more with *source_file*, and go on to any remaining files.

iii. If *source_file* is a file of type symbolic link, the pathname contained in *dest_file* shall be the same as the pathname contained in *source_file*.

If this fails for any reason, *cp* shall write a diagnostic message to standard error, do nothing more with *source_file*, and go on to any remaining files.

If the implementation provides additional or alternate access control mechanisms (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 4.4, File Access Permissions), their effect on copies of files is implementation-defined.

OPTIONS

The *cp* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–f If a file descriptor for a destination file cannot be obtained, as described in step 3.a.ii., attempt to unlink the destination file and proceed.

–H Take actions based on the type and contents of the file referenced by any symbolic link specified as a *source_file* operand.

–i Write a prompt to standard error before copying to any existing destination file. If the response from the standard input is affirmative, the copy shall be attempted;

otherwise, it shall not.

–L Take actions based on the type and contents of the file referenced by any symbolic link specified as a *source_file* operand or any symbolic links encountered during traversal of a file hierarchy.

–P Take actions on any symbolic link specified as a *source_file* operand or any symbolic link encountered during traversal of a file hierarchy.

–p Duplicate the following characteristics of each source file in the corresponding destination file:

1. The time of last data modification and time of last access. If this duplication fails for any reason, *cp* shall write a diagnostic message to standard error.
2. The user ID and group ID. If this duplication fails for any reason, it is unspecified whether *cp* writes a diagnostic message to standard error.
3. The file permission bits and the S_ISUID and S_ISGID bits. Other, implementation-defined, bits may be duplicated as well. If this duplication fails for any reason, *cp* shall write a diagnostic message to standard error.

If the user ID or the group ID cannot be duplicated, the file permission bits S_ISUID and S_ISGID shall be cleared. If these bits are present in the source file but are not duplicated in the destination file, it is unspecified whether *cp* writes a diagnostic message to standard error.

The order in which the preceding characteristics are duplicated is unspecified. The *dest_file* shall not be deleted if these characteristics cannot be preserved.

–R Copy file hierarchies.

OB **–r** Copy file hierarchies. The treatment of special files is implementation-defined.

Specifying more than one of the mutually-exclusive options **–H**, **–L**, and **–P** shall not be considered an error. The last option specified shall determine the behavior of the utility.

OPERANDS

The following operands shall be supported:

source_file A pathname of a file to be copied.

target_file A pathname of an existing or nonexistent file, used for the output when a single file is copied.

target A pathname of a directory to contain the copied files.

STDIN

The standard input shall be used to read an input line in response to each prompt specified in the STDERR section. Otherwise, the standard input shall not be used.

INPUT FILES

The input files specified as operands may be of any file type.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *cp*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements used in the extended regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES* category.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes used in the extended regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES* category.

LC_MESSAGES
Determine the locale for the processing of affirmative responses that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
Not used.

STDERR
A prompt shall be written to standard error under the conditions specified in the DESCRIPTION section. The prompt shall contain the destination pathname, but its format is otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic messages.

OUTPUT FILES
The output files may be of any type.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 All files were copied successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS
If *cp* is prematurely terminated by a signal or error, files or file hierarchies may be only partially copied and files and directories may have incorrect permissions or access and modification times.

APPLICATION USAGE

The difference between **–R** and **–r** is in the treatment by *cp* of file types other than regular and directory. The original **–r** flag, for historic reasons, does not handle special files any differently from regular files, but always reads the file and copies its contents. This has obvious problems in the presence of special file types; for example, character devices, FIFOs, and sockets. The **–R** option is intended to recreate the file hierarchy and the **–r** option supports historical practice. It was anticipated that a future version of this volume of IEEE Std 1003.1-200x would deprecate the **–r** option, and for that reason, there has been no attempt to fix its behavior with respect to FIFOs or other file types where copying the file is clearly wrong. However, some implementations support **–r** with the same abilities as the **–R** defined in this volume of IEEE Std 1003.1-200x. To accommodate them as well as systems that do not, the differences between **–r** and **–R** are implementation-defined. Implementations may make them identical. The **–r** option is now marked obsolescent.

The set-user-ID and set-group-ID bits are explicitly cleared when files are created. This is to prevent users from creating programs that are set-user-ID or set-group-ID to them when copying files or to make set-user-ID or set-group-ID files accessible to new groups of users. For example, if a file is set-user-ID and the copy has a different group ID than the source, a new group of users has execute permission to a set-user-ID program than did previously. In particular, this is a problem for superusers copying users' trees.

EXAMPLES

None.

RATIONALE

The **–i** option exists on BSD systems, giving applications and users a way to avoid accidentally removing files when copying. Although the 4.3 BSD version does not prompt if the standard input is not a terminal, the standard developers decided that use of **–i** is a request for interaction, so when the destination path exists, the utility takes instructions from whatever responds on standard input.

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified because implementations may desire more descriptive prompts than those used on historical implementations. Therefore, an application using the **–i** option relies on the system to provide the most suitable dialog directly with the user, based on the behavior specified.

The **–p** option is historical practice on BSD systems, duplicating the time of last data modification and time of last access. This volume of IEEE Std 1003.1-200x extends it to preserve the user and group IDs, as well as the file permissions. This requirement has obvious problems in that the directories are almost certainly modified after being copied. This volume of IEEE Std 1003.1-200x requires that the modification times be preserved. The statement that the order in which the characteristics are duplicated is unspecified is to permit implementations to provide the maximum amount of security for the user. Implementations should take into account the obvious security issues involved in setting the owner, group, and mode in the wrong order or creating files with an owner, group, or mode different from the final value.

It is unspecified whether *cp* writes diagnostic messages when the user and group IDs cannot be set due to the widespread practice of users using **–p** to duplicate some portion of the file characteristics, indifferent to the duplication of others. Historic implementations only write diagnostic messages on errors other than [EPERM].

The **–r** option is historical practice on BSD and BSD-derived systems, copying file hierarchies as opposed to single files. This functionality is used heavily in historical applications, and its loss would significantly decrease consensus. The **–R** option was added as a close synonym to the **–r** option, selected for consistency with all other options in this volume of IEEE Std 1003.1-200x

that do recursive directory descent.

When a failure occurs during the copying of a file hierarchy, *cp* is required to attempt to copy files that are on the same level in the hierarchy or above the file where the failure occurred. It is unspecified if *cp* shall attempt to copy files below the file where the failure occurred (which cannot succeed in any case).

Permissions, owners, and groups of created special file types have been deliberately left as implementation-defined. This is to allow systems to satisfy special requirements (for example, allowing users to create character special devices, but requiring them to be owned by a certain group). In general, it is strongly suggested that the permissions, owner, and group be the same as if the user had run the historical *mknod*, *ln*, or other utility to create the file. It is also probable that additional privileges are required to create block, character, or other implementation-defined special file types.

Additionally, the **–p** option explicitly requires that all set-user-ID and set-group-ID permissions be discarded if any of the owner or group IDs cannot be set. This is to keep users from unintentionally giving away special privilege when copying programs.

When creating regular files, historical versions of *cp* use the mode of the source file as modified by the file mode creation mask. Other choices would have been to use the mode of the source file unmodified by the creation mask or to use the same mode as would be given to a new file created by the user (plus the execution bits of the source file) and then modify it by the file mode creation mask. In the absence of any strong reason to change historic practice, it was in large part retained.

When creating directories, historical versions of *cp* use the mode of the source directory, plus read, write, and search bits for the owner, as modified by the file mode creation mask. This is done so that *cp* can copy trees where the user has read permission, but the owner does not. A side effect is that if the file creation mask denies the owner permissions, *cp* fails. Also, once the copy is done, historical versions of *cp* set the permissions on the created directory to be the same as the source directory, unmodified by the file creation mask.

This behavior has been modified so that *cp* is always able to create the contents of the directory, regardless of the file creation mask. After the copy is done, the permissions are set to be the same as the source directory, as modified by the file creation mask. This latter change from historical behavior is to prevent users from accidentally creating directories with permissions beyond those they would normally set and for consistency with the behavior of *cp* in creating files.

It is not a requirement that *cp* detect attempts to copy a file to itself; however, implementations are strongly encouraged to do so. Historical implementations have detected the attempt in most cases.

There are two methods of copying subtrees in this volume of IEEE Std 1003.1-200x. The other method is described as part of the *pax* utility (see *pax* (on page 2900)). Both methods are historical practice. The *cp* utility provides a simpler, more intuitive interface, while *pax* offers a finer granularity of control. Each provides additional functionality to the other; in particular, *pax* maintains the hard-link structure of the hierarchy, while *cp* does not. It is the intention of the standard developers that the results be similar (using appropriate option combinations in both utilities). The results are not required to be identical; there seemed insufficient gain to applications to balance the difficulty of implementations having to guarantee that the results would be exactly identical.

The wording allowing *cp* to copy a directory to implementation-defined file types not specified by the System Interfaces volume of IEEE Std 1003.1-200x is provided so that implementations supporting symbolic links are not required to prohibit copying directories to symbolic links. Other extensions to the System Interfaces volume of IEEE Std 1003.1-200x file types may need to

use this loophole as well.

FUTURE DIRECTIONS

The **–r** option may be removed; use **–R** instead.

SEE ALSO

mv, *find*, *ln*, *pax*

CHANGE HISTORY

First released in Issue 2.

Issue 6

The **–r** option is marked obsolescent.

The new options **–H**, **–L**, and **–P** are added to align with the IEEE P1003.2b draft standard. These options affect the processing of symbolic links. |

IEEE PASC Interpretation 1003.2 #194 is applied, adding a description of the **–P** option. |

10586 **NAME**

10587 crontab — schedule periodic background work

10588 **SYNOPSIS**

10589 UP
```
crontab [file]
```

10590
```
crontab [ –e | –l | –r ]
```

10591

10592 **DESCRIPTION**

10593 The *crontab* utility shall create, replace, or edit a user's *crontab* entry; a crontab entry is a list of
10594 commands and the times at which they shall be executed. The new crontab entry can be input by
10595 specifying *file* or input from standard input if no *file* operand is specified, or by using an editor, if
10596 **–e** is specified.

10597 Upon execution of a command from a crontab entry, the implementation shall supply a default
10598 environment, defining at least the following environment variables:

10599 *HOME* A pathname of the user's home directory.

10600 *LOGNAME* The user's login name.

10601 *PATH* A string representing a search path guaranteed to find all of the standard utilities.

10602 *SHELL* A pathname of the command interpreter. When *crontab* is invoked as specified by
10603 this volume of IEEE Std 1003.1-200x, the value shall be a pathname for *sh*.

10604 The values of these variables when *crontab* is invoked as specified by this volume of
10605 IEEE Std 1003.1-200x shall not affect the default values provided when the scheduled command
10606 is run.

10607 If standard output and standard error are not redirected by commands executed from the
10608 crontab entry, any generated output or errors shall be mailed, via an implementation-defined
10609 method, to the user.

10610 XSI Users shall be permitted to use *crontab* if their names appear in the file **/usr/lib/cron/cron.allow**. |
10611 If that file does not exist, the file **/usr/lib/cron/cron.deny** shall be checked to determine whether |
10612 the user shall be denied access to *crontab*. If neither file exists, only a process with appropriate |
10613 privileges shall be allowed to submit a job. If only **cron.deny** exists and is empty, global usage |
10614 shall be permitted. The **cron.allow** and **cron.deny** files shall consist of one user name per line. |

10615 **OPTIONS**

10616 The *crontab* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
10617 12.2, Utility Syntax Guidelines.

10618 The following options shall be supported:

10619 **–e** Edit a copy of the invoking user's crontab entry, or create an empty entry to edit if
10620 the crontab entry does not exist. When editing is complete, the entry shall be
10621 installed as the user's crontab entry.

10622 **–l** (The letter ell.) List the invoking user's crontab entry.

10623 **–r** Remove the invoking user's crontab entry.

10624 **OPERANDS**

10625 The following operand shall be supported:

10626 *file* The pathname of a file that contains specifications, in the format defined in the
10627 INPUT FILES section, for crontab entries.

10628 **STDIN**

10629 See the INPUT FILES section.

10630 **INPUT FILES**

10631 In the POSIX locale, the user or application shall ensure that a crontab entry is a text file
10632 consisting of lines of six fields each. The fields shall be separated by <blank>s. The first five
10633 fields shall be integer patterns that specify the following:

10634 1. Minute [0,59]
10635 2. Hour [0,23]
10636 3. Day of the month [1,31]
10637 4. Month of the year [1,12]
10638 5. Day of the week ([0,6] with 0=Sunday)

10639 Each of these patterns can be either an asterisk (meaning all valid values), an element, or a list of
10640 elements separated by commas. An element shall be either a number or two numbers separated
10641 by a hyphen (meaning an inclusive range). The specification of days can be made by two fields
10642 (day of the month and day of the week). If month, day of month, and day of week are all
10643 asterisks, every day shall be matched. If either the month or day of month is specified as an
10644 element or list, but the day of week is an asterisk, the month and day of month fields shall
10645 specify the days that match. If both month and day of month are specified as asterisk, but day of
10646 week is an element or list, then only the specified days of the week match. Finally, if either the
10647 month or day of month is specified as an element or list, and the day of week is also specified as
10648 an element or list, then any day matching either the month and day of month, or the day of
10649 week, shall be matched.

10650 The sixth field of a line in a crontab entry is a string that shall be executed by *sh* at the specified
10651 times. A percent sign character in this field shall be translated to a <newline>. Any character
10652 preceded by a backslash (including the '%') shall cause that character to be treated literally.
10653 Only the first line (up to a '%' or end-of-line) of the command field shall be executed by the
10654 command interpreter. The other lines shall be made available to the command as standard input.

10655 Blank lines and those whose first non-<blank> is '#' shall be ignored.

10656 XSI The text files **/usr/lib/cron/cron.allow** and **/usr/lib/cron/cron.deny** shall contain zero or more
10657 user names, one per line, of users who are, respectively, authorized or denied access to the
10658 service underlying the *crontab* utility.

10659 **ENVIRONMENT VARIABLES**

10660 The following environment variables shall affect the execution of *crontab*:

10661 *EDITOR* Determine the editor to be invoked when the **–e** option is specified. The default
10662 editor shall be *vi*.

10663 *LANG* Provide a default value for the internationalization variables that are unset or null.
10664 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
10665 Internationalization Variables for the precedence of internationalization variables
10666 used to determine the values of locale categories.)

10667 *LC_ALL* If set to a non-empty string value, override the values of all the other
10668 internationalization variables.

10669 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
10670 characters (for example, single-byte as opposed to multi-byte characters in
10671 arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
If the **–l** option is specified, the crontab entry shall be written to the standard output.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS
The user's crontab entry is not submitted, removed, edited, or listed.

APPLICATION USAGE
The format of the *crontab* entry shown here is guaranteed only for the POSIX locale. Other cultures may be supported with substantially different interfaces, although implementations are encouraged to provide comparable levels of functionality.

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 volume of IEEE Std 1003.1-200x'' 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.

A typical user error is to type only *crontab*; this causes the system to wait for the new crontab entry on standard input. If end-of-file is typed (generally <control>-D), the crontab entry is replaced by an empty file. In this case, the user should type the interrupt character, which prevents the crontab entry from being replaced.

EXAMPLES
1. Clean up **core** files every weekday morning at 3:15 am:

```
15 3 * * 1-5 find $HOME –name core 2>/dev/null | xargs rm –f
```

2. Mail a birthday greeting:

```
0 12 14 2 * mailx john%Happy Birthday!%Time for lunch.
```

3. 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.

RATIONALE

All references to a *cron* daemon and to *cron files* have been omitted. 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 a similar mechanism) use this format internally. For example, systems derived from BSD are likely to 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.

FUTURE DIRECTIONS

None.

SEE ALSO

at

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

10737 **NAME**

10738 csplit — split files based on context

10739 **SYNOPSIS**

10740 UP
```
csplit [–ks][–f prefix][–n number] file arg1 ...argn
```
10741

10742 **DESCRIPTION**

10743 The *csplit* utility shall read the file named by the *file* operand, write all or part of that file into
10744 other files as directed by the *arg* operands, and write the sizes of the files.

10745 **OPTIONS**

10746 The *csplit* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
10747 12.2, Utility Syntax Guidelines.

10748 The following options shall be supported:

10749 **–f** *prefix* Name the created files *prefix***00**, *prefix***01**, ..., *prefix*n. The default is **xx00** ... **xx***n*. If
10750 the *prefix* argument would create a filename exceeding {NAME_MAX} bytes, an
10751 error shall result, *csplit* shall exit with a diagnostic message and no files shall be
10752 created.

10753 **–k** Leave previously created files intact. By default, *csplit* shall remove created files if
10754 an error occurs.

10755 **–n** *number* Use *number* decimal digits to form filenames for the file pieces. The default shall be
10756 2.

10757 **–s** Suppress the output of file size messages.

10758 **OPERANDS**

10759 The following operands shall be supported:

10760 *file* The pathname of a text file to be split. If *file* is '–', the standard input shall be
10761 used.

10762 The operands *arg1* ... *argn* can be a combination of the following:

10763 /*rexp*/**[***offset***]**
10764 A file shall be created using the content of the lines from the current line up to, but
10765 not including, the line that results from the evaluation of the regular expression
10766 with *offset*, if any, applied. The regular expression *rexp* shall follow the rules for
10767 basic regular expressions described in the Base Definitions volume of
10768 IEEE Std 1003.1-200x, Section 9.3, Basic Regular Expressions. The application shall
10769 use the sequence "\/" to specify a slash character within the *rexp*. The optional
10770 offset shall be a positive or negative integer value representing a number of lines.
10771 A positive integer value can be preceded by '+'. If the selection of lines from an
10772 *offset* expression of this type would create a file with zero lines, or one with greater
10773 than the number of lines left in the input file, the results are unspecified. After the
10774 section is created, the current line shall be set to the line that results from the
10775 evaluation of the regular expression with any offset applied. If the current line is
10776 the first line in the file and a regular expression operation has not yet been
10777 performed, the pattern match of *rexp* shall be applied from the current line to the
10778 end of the file. Otherwise, the pattern match of *rexp* shall be applied from the line
10779 following the current line to the end of the file.

10780 %*rexp*%**[***offset***]**
10781 Equivalent to /*rexp*/**[***offset***]**, except that no file shall be created for the selected
10782 section of the input file. The application shall use the sequence "\%" to specify a

percent-sign character within the *rexp*.

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.

STDIN

See the INPUT FILES section.

INPUT FILES

The input file shall be a text file.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *csplit*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes within regular expressions.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

If the **–k** option is specified, created files shall be retained. Otherwise, the default action occurs.

STDOUT

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>
```

STDERR

The standard error shall be used only for diagnostic messages.

10824 **OUTPUT FILES**

10825 The output files shall contain portions of the original input file; otherwise, unchanged.

10826 **EXTENDED DESCRIPTION**

10827 None.

10828 **EXIT STATUS**

10829 The following exit values shall be returned:

10830 0 Successful completion.

10831 >0 An error occurred.

10832 **CONSEQUENCES OF ERRORS**

10833 By default, created files shall be removed if an error occurs. When the **–k** option is specified,
10834 created files shall not be removed if an error occurs.

10835 **APPLICATION USAGE**

10836 None.

10837 **EXAMPLES**

10838 1. This example creates four files, **cobol00** ... **cobol03**:

10839 ```
csplit –f cobol file '/procedure division/' /par5./ /par16./
```

10840 After editing the split files, they can be recombined as follows:

10841 ```
cat cobol0[0–3] > file
```

10842 Note that this example overwrites the original file.

10843 2. This example would split the file after the first 99 lines, and every 100 lines thereafter, up
10844 to 9 999 lines; this is because lines in the file are numbered from 1 rather than zero, for
10845 historical reasons:

10846 ```
csplit –k file 100 {99}
```

10847 3. Assuming that **prog.c** follows the C-language coding convention of ending routines with a
10848 `'}'` at the beginning of the line, this example creates a file containing each separate C
10849 routine (up to 21) in **prog.c**:

10850 ```
csplit –k prog.c '%main(%'  '/^}/+1' {20}
```

10851 **RATIONALE**

10852 The **–n** option was added to extend the range of filenames that could be handled.

10853 Consideration was given to adding a **–a** flag to use the alphabetic filename generation used by
10854 the historical *split* utility, but the functionality added by the **–n** option was deemed to make
10855 alphabetic naming unnecessary.

10856 **FUTURE DIRECTIONS**

10857 None.

10858 **SEE ALSO**

10859 *sed*, *split*

10860 **CHANGE HISTORY**

10861 First released in Issue 2.

Issue 5

FUTURE DIRECTIONS section added.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The APPLICATION USAGE section is added.

The description of regular expression operands is changed to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

10870 **NAME**

10871 ctags — create a tags file (**DEVELOPMENT**, **FORTRAN**)

10872 **SYNOPSIS**

10873 UP
```
ctags [–a][–f tagsfile] pathname ...
```

10874
```
ctags –x pathname ...
```

10875

10876 **DESCRIPTION**

10877 The *ctags* utility shall be provided on systems that support the User Portability Utilities option,
10878 the Software Development Utilities option, and either or both of the C-Language Development
10879 Utilities option and FORTRAN Development Utilities option. On other systems, it is optional.

10880 The *ctags* utility shall write a *tags* file or an index of objects from C-language or FORTRAN
10881 source files specified by the *pathname* operands. The tags file shall list the locators of language-
10882 specific objects within the source files. A locator consists of a name, pathname, and either a
10883 search pattern or a line number that can be used in searching for the object definition. The
10884 objects that shall be recognized are specified in the EXTENDED DESCRIPTION section.

10885 **OPTIONS**

10886 The *ctags* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
10887 12.2, Utility Syntax Guidelines.

10888 The following options shall be supported:

10889 **–a** Append to tags file.

10890 **–f** *tagsfile* Write the object locator lists into *tagsfile* instead of the default file named **tags** in
10891 the current directory.

10892 **–x** Produce a list of object names, the line number, and filename in which each is
10893 defined, as well as the text of that line, and write this to the standard output. A
10894 **tags** file shall not be created when **–x** is specified.

10895 **OPERANDS**

10896 The following *pathname* operands are supported:

10897 *file***.c** Files with basenames ending with the **.c** suffix shall be treated as C-language
10898 source code. Such files that are not valid input to *c99* produce unspecified results.

10899 *file***.h** Files with basenames ending with the **.h** suffix shall be treated as C-language
10900 source code. Such files that are not valid input to *c99* produce unspecified results.

10901 *file***.f** Files with basenames ending with the **.f** suffix shall be treated as FORTRAN-
10902 language source code. Such files that are not valid input to *fort77* produce
10903 unspecified results.

10904 The handling of other files is implementation-defined.

10905 **STDIN**

10906 See the INPUT FILES section.

10907 **INPUT FILES**

10908 The input files shall be text files containing source code in the language indicated by the operand
10909 filename suffixes.

10910 **ENVIRONMENT VARIABLES**

10911 The following environment variables shall affect the execution of *ctags*:

10912 *LANG* Provide a default value for the internationalization variables that are unset or null.
10913 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
10914 Internationalization Variables for the precedence of internationalization variables
10915 used to determine the values of locale categories.)

10916 *LC_ALL* If set to a non-empty string value, override the values of all the other
10917 internationalization variables.

10918 *LC_COLLATE*
10919 Determine the order in which output is sorted for the **–x** option. The POSIX locale
10920 determines the order in which the tags file is written.

10921 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
10922 characters (for example, single-byte as opposed to multi-byte characters in
10923 arguments and input files). When processing C-language source code, if the locale
10924 is not compatible with the C locale described by the ISO C standard, the results are
10925 unspecified.

10926 *LC_MESSAGES*
10927 Determine the locale that should be used to affect the format and contents of
10928 diagnostic messages written to standard error.

10929 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

10930 **ASYNCHRONOUS EVENTS**

10931 Default.

10932 **STDOUT**

10933 The list of object name information produced by the **–x** option shall be written to standard
10934 output in the following format:

10935 `"%s %d %s %s", <object-name>, <line-number>, <filename>,`
10936 `    <text>`

10937 where <*text*> is the text of line <*line-number*> of file <*filename*>.

10938 **STDERR**

10939 The standard error shall be used only for diagnostic messages.

10940 **OUTPUT FILES**

10941 When the **–x** option is not specified, the format of the output file shall be:

10942 `"%s\t%s\t/%s/\n", <identifier>, <filename>, <pattern>`

10943 where <*pattern*> is a search pattern that could be used by an editor to find the defining instance
10944 of <*identifier*> in <*filename*> (where *defining instance* is indicated by the declarations listed in the
10945 EXTENDED DESCRIPTION).

10946 An optional circumflex (`'^'`) can be added as a prefix to <*pattern*>, and an optional dollar sign
10947 can be appended to <*pattern*> to indicate that the pattern is anchored to the beginning (end) of a
10948 line of text. Any slash or backslash characters in <*pattern*> shall be preceded by a backslash
10949 character. The anchoring circumflex, dollar sign, and escaping backslash characters shall not be
10950 considered part of the search pattern. All other characters in the search pattern shall be
10951 considered literal characters.

10952 An alternative format is:

"%s\t%s\t?%s?\n", <*identifier*>, <*filename*>, <*pattern*>

which is identical to the first format except that slashes in <*pattern*> shall not be preceded by escaping backslash characters, and question mark characters in <*pattern*> shall be preceded by backslash characters.

A second alternative format is:

"%s\t%s\t%d\n", <*identifier*>, <*filename*>, <*lineno*>

where <*lineno*> is a decimal line number that could be used by an editor to find <*identifier*> in <*filename*>.

Neither alternative format shall be produced by *ctags* when it is used as described by IEEE Std 1003.1-200x, but the standard utilities that process tags files shall be able to process those formats as well as the first format.

In any of these formats, the file shall be sorted by identifier, based on the collation sequence in the POSIX locale.

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
- Type definitions
- Macros with arguments

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

- Function prototypes
- Structures
- Unions
- Global variable definitions
- Enumeration types
- 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, *ctags* produces undefined results for C-language source code files. It should write to standard error a message identifying this condition and cause a non-zero 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, *ctags* produces unspecified results for FORTRAN source code files. It should write to standard error a message identifying this condition and cause a non-zero exit status to be produced.

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

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

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 locale as that in effect when *ctags* **–x** is run, results might not be as expected.

The description of C-language processing says ‘‘attempts to’’ because the C language can be greatly confused, especially through the use of **#define**s, and this utility would be of no use if the real C preprocessor were run to identify them. The output from *ctags* may be fooled and incorrect for various constructs.

EXAMPLES

None.

RATIONALE

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 early proposal 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 text for *LC_CTYPE* about compatibility with the C locale acknowledges that the ISO C standard imposes requirements on the locale used to process C source. This could easily be a superset of that known as ‘‘the C locale’’ by way of implementation extensions, or one of a few alternative locales for systems supporting different codesets. No statement is made for FORTRAN because the ANSI X3.9-1978 standard (FORTRAN 77) does not (yet) define a similar locale concept. However, a general rule in this volume of IEEE Std 1003.1-200x is that any time that locales do not match (preparing a file for one locale and processing it in another), the results are suspect.

The collation sequence of the tags file is not affected by *LC_COLLATE* because it is typically not used by human readers, but only by programs such as *vi* to locate the tag within the source files. Using the POSIX locale eliminates some of the problems of coordinating locales between the *ctags* file creator and the *vi* file reader.

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 patrerns or line numbers to find the identifiers because the historical *vi* recognizes either.

The *ctags* utility does not produce the format using line numbers because it is not useful following any source file changes that add or delete lines. The documented search patterns match historical practice. It should be noted that literal leading circumflex or trailing dollar-sign characters in the search pattern will only behave correctly if anchored to the beginning of the line or end of the line by an additional circumflex or dollar-sign character.

Historical implementations also understand the objects used by the languages Pascal and sometimes LISP, and they 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 volume of IEEE Std 1003.1-200x:

–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, filename, and page number (assuming 64-line pages). Since the output is 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
```

The special treatment of the tag **main** makes the use of *ctags* practical in directories with more than one program.

FUTURE DIRECTIONS

None.

SEE ALSO

c99, *fort77*, *vi*

CHANGE HISTORY

First released in Issue 4.

Issue 5

FUTURE DIRECTIONS section added.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The OUTPUT FILES section is changed to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

IEEE PASC Interpretation 1003.2 #168 is applied, changing ‘‘create’’ to ‘‘write’’ in the DESCRIPTION.

NAME

cut — cut out selected fields of each line of a file

SYNOPSIS

```
cut –b list [–n] [file ...]
cut –c list [file ...]
cut –f list [–d delim][–s][file ...]
```

DESCRIPTION

The *cut* utility shall cut out bytes (**–b** option), characters (**–c** option) or character-delimited fields (**–f** option) from each line in one or more files, concatenate them, and write them to standard output.

OPTIONS

The *cut* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The application shall ensure that the option-argument *list* (see options **–b**, **–c**, and **–f** below) is a comma-separated list or <blank>-separated list of positive numbers and ranges. Ranges can be in three forms. The first is two positive numbers separated by a hyphen (*low–high*), which represents all fields from the first number to the second number. The second is a positive number preceded by a hyphen (*–high*), which represents all fields from field number 1 to that number. The third is a positive number followed by a hyphen (*low–*), which represents that number to the last field, inclusive. The elements in *list* can be repeated, can overlap, and can be specified in any order, but the bytes, characters, or fields selected shall be written in the order of the input data. If an element appears in the selection list more than once, it shall be written exactly once.

The following options shall be supported:

–b *list* Cut based on a *list* of bytes. Each selected byte shall be output unless the **–n** option is also specified. It shall not be an error to select bytes not present in the input line.

–c *list* Cut based on a *list* of characters. Each selected character shall be output. It shall not be an error to select characters not present in the input line.

–d *delim* Set the field delimiter to the character *delim*. The default is the <tab>.

–f *list* Cut based on a *list* of fields, assumed to be separated in the file by a delimiter character (see **–d**). Each selected field shall be output. Output fields shall be separated by a single occurrence of the field delimiter character. Lines with no field delimiters shall be passed through intact, unless **–s** is specified. It shall not be an error to select fields not present in the input line.

–n Do not split characters. When specified with the **–b** option, each element in *list* of the form *low–high* (hyphen-separated numbers) shall be modified as follows:

- If the byte selected by *low* is not the first byte of a character, *low* shall be decremented to select the first byte of the character originally selected by *low*. If the byte selected by *high* is not the last byte of a character, *high* shall be decremented to select the last byte of the character prior to the character originally selected by *high*, or zero if there is no prior character. If the resulting range element has *high* equal to zero or *low* greater than *high*, the list element shall be dropped from *list* for that input line without causing an error.

Each element in *list* of the form *low–* shall be treated as above with *high* set to the number of bytes in the current line, not including the terminating <newline>. Each

element in *list* of the form –*high* shall be treated as above with *low* set to 1. Each element in *list* of the form *num* (a single number) shall be treated as above with *low* set to *num* and *high* set to *num*.

–s Suppress lines with no delimiter characters, when used with the **–f** option. Unless specified, lines with no delimiters shall be passed through untouched.

OPERANDS

The following operand shall be supported:

file A pathname of an input file. If no *file* operands are specified, or if a *file* operand is `'–'`, the standard input shall be used.

STDIN

The standard input shall be used only if no *file* operands are specified, or if a *file* operand is `'–'`. See the INPUT FILES section.

INPUT FILES

The input files shall be text files, except that line lengths shall be unlimited.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *cut*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The *cut* utility output shall be a concatenation of the selected bytes, characters, or fields (one of the following):

```
"%s\n", <concatenation of bytes>
"%s\n", <concatenation of characters>
"%s\n", <concatenation of fields and field delimiters>
```

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All input files were output successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Earlier versions of the *cut* utility worked in an environment where bytes and characters were considered equivalent (modulo <backspace> and <tab> processing in some implementations). In the extended world of multi-byte characters, the new **–b** option has been added. The **–n** option (used with **–b**) allows it to be used to act on bytes rounded to character boundaries. The algorithm specified for **–n** guarantees that:

```
cut –b 1–500 –n file > file1
cut –b 501– –n file > file2
```

ends up with all the characters in **file** appearing exactly once in **file1** or **file2**. (There is, however, a <newline> in both **file1** and **file2** for each <newline> in **file**.)

EXAMPLES

Examples of the option qualifier list:

1,4,7 Select the first, fourth, and seventh bytes, characters, or fields and field delimiters.

1–3,8 Equivalent to 1,2,3,8.

–5,10 Equivalent to 1,2,3,4,5,10.

3– Equivalent to third to last, inclusive.

The *low–high* forms are not always equivalent when used with **–b** and **–n** and multi-byte characters; see the description of **–n**.

The following command:

```
cut –d : –f 1,6 /etc/passwd
```

reads the System V password file (user database) and produces lines of the form:

```
<user ID>:<home directory>
```

Most utilities in this volume of IEEE Std 1003.1-200x work on text files. The *cut* utility can be used to turn files with arbitrary line lengths into a set of text files containing the same data. The *paste* utility can be used to create (or recreate) files with arbitrary line lengths. For example, if **file** contains long lines:

```
cut –b 1–500 –n file > file1
cut –b 501– –n file > file2
```

creates **file1** (a text file) with lines no longer than 500 bytes (plus the <newline>) and **file2** that contains the remainder of the data from **file**. (Note that **file2** is not a text file if there are lines in **file** that are longer than 500 + {LINE_MAX} bytes.) The original file can be recreated from **file1** and **file2** using the command:

```
paste –d "\0" file1 file2 > file
```

RATIONALE

Some historical implementations do not count <backspace>s in determining character counts with the **–c** option. This may be useful for using *cut* for processing *nroff* output. It was deliberately decided not to have the **–c** option treat either <backspace>s or <tab>s in any special fashion. The *fold* utility does treat these characters specially.

Unlike other utilities, some historical implementations of *cut* exit after not finding an input file, rather than continuing to process the remaining *file* operands. This behavior is prohibited by this volume of IEEE Std 1003.1-200x, where only the exit status is affected by this problem.

The behavior of *cut* when provided with either mutually-exclusive options or options that do not work logically together has been deliberately left unspecified in favor of global wording in Section 1.11 (on page 2221).

The OPTIONS section was changed in response to P1003.2-N149. The change represents historical practice on all known systems. The original standard was ambiguous on the nature of the output.

The *list* option-arguments are historically used to select the portions of the line to be written, but do not affect the order of the data. For example:

```
echo abcdefghi | cut –c6,2,4–7,1
```

yields `"abdefg"`.

A proposal to enhance *cut* with the following option:

–o Preserve the selected field order. When this option is specified, each byte, character, or field (or ranges of such) shall be written in the order specified by the *list* option-argument, even if this requires multiple outputs of the same bytes, characters, or fields.

was rejected because this type of enhancement is outside the scope of the IEEE P1003.2b draft standard.

FUTURE DIRECTIONS

None.

SEE ALSO

grep, *paste*, Section 2.5 (on page 2235)

CHANGE HISTORY

First released in Issue 2.

Issue 6

The OPTIONS section is changed to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

11228 **NAME**

11229 cxref — generate a C-language program cross-reference table (**DEVELOPMENT**)

11230 **SYNOPSIS**

11231 XSI
```
cxref [–cs][–o file][–w num] [–D name[=def]]...[–I dir]...
    [–U name]... file ...
```
11232

11233

11234 **DESCRIPTION**

11235 The *cxref* utility shall analyze a collection of C-language *file*s and attempt to build a cross-
11236 reference table. Information from **#define** lines shall be included in the symbol table. A sorted
11237 listing shall be written to standard output of all symbols (auto, static, and global) in each *file*
11238 separately, or with the **–c** option, in combination. Each symbol shall contain an asterisk before
11239 the declaring reference.

11240 **OPTIONS**

11241 The *cxref* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
11242 12.2, Utility Syntax Guidelines, except that the order of the **–D**, **–I**, and **–U** options (which are
11243 identical to their interpretation by *c99*) is significant. The following options shall be supported:

11244 **–c** Write a combined cross-reference of all input files.

11245 **–s** Operate silently; do not print input filenames.

11246 **–o** *file* Direct output to named *file*.

11247 **–w** *num* Format output no wider than *num* (decimal) columns. This option defaults to 80 if
11248 *num* is not specified or is less than 51.

11249 **–D** Equivalent to *c99*.

11250 **–I** Equivalent to *c99*.

11251 **–U** Equivalent to *c99*.

11252 **OPERANDS**

11253 The following operand shall be supported:

11254 *file* A pathname of a C-language source file.

11255 **STDIN**

11256 Not used.

11257 **INPUT FILES**

11258 The input files are C-language source files.

11259 **ENVIRONMENT VARIABLES**

11260 The following environment variables shall affect the execution of *cxref*:

11261 *LANG* Provide a default value for the internationalization variables that are unset or null.
11262 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
11263 Internationalization Variables for the precedence of internationalization variables
11264 used to determine the values of locale categories.)

11265 *LC_ALL* If set to a non-empty string value, override the values of all the other
11266 internationalization variables.

11267 *LC_COLLATE*
11268 Determine the locale for the ordering of the output.

11269 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
11270 characters (for example, single-byte as opposed to multi-byte characters in

arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
The standard output shall be used for the cross-reference listing, unless the **−o** option is used to select a different output file.

The format of standard output is unspecified, except that the following information shall be included:

- If the **−c** option is not specified, each portion of the listing shall start with the name of the input file on a separate line.
- The name line shall be followed by a sorted list of symbols, each with its associated location pathname, the name of the function in which it appears (if it is not a function name itself), and line number references.
- Each line number may be preceded by an asterisk (`'*'`) flag, meaning that this is the declaring reference. Other single-character flags, with implementation-defined meanings, may be included.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
The output file named by the **−o** option shall be used instead of standard output.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
None.

EXAMPLES
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

11311 **SEE ALSO**

11312 *c99*

11313 **CHANGE HISTORY**

11314 First released in Issue 2.

11315 **Issue 5**

11316 In the SYNOPSIS, [**–U** *dir*] is changed to [**–U** *name*].

11317 **Issue 6**

11318 The APPLICATION USAGE section is added.

11319 **NAME**

11320 date — write the date and time

11321 **SYNOPSIS**

11322 `date [–u] [+`*format*`]`

11323 XSI `date [–u] `*mmddhhmm*`[[`*cc*`]`*yy*`]`

11324

11325 **DESCRIPTION**

11326 XSI The *date* utility shall write the date and time to standard output or attempt to set the system date
11327 and time. By default, the current date and time shall be written. If an operand beginning with
11328 '+' is specified, the output format of *date* shall be controlled by the conversion specifications
11329 and other text in the operand.

11330 **OPTIONS**

11331 The *date* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
11332 12.2, Utility Syntax Guidelines.

11333 The following option shall be supported:

11334 **–u** Perform operations as if the *TZ* environment variable was set to the string "UTC0",
11335 or its equivalent historical value of "GMT0". Otherwise, *date* shall use the
11336 timezone indicated by the *TZ* environment variable or the system default if that
11337 variable is unset or null.

11338 **OPERANDS**

11339 The following operands shall be supported:

11340 *+format* When the format is specified, each conversion specifier shall be replaced in the
11341 standard output by its corresponding value. All other characters shall be copied to
11342 the output without change. The output always shall be terminated with a
11343 <newline>.

11344 **Conversion Specifications**

11345 %a Locale's abbreviated weekday name.

11346 %A Locale's full weekday name.

11347 %b Locale's abbreviated month name.

11348 %B Locale's full month name.

11349 %c Locale's appropriate date and time representation.

11350 %C Century (a year divided by 100 and truncated to an integer) as a decimal |
11351 number [00,99]. |

11352 %d Day of the month as a decimal number [01,31]. |

11353 %D Date in the format *mm*/*dd*/*yy*.

11354 %e Day of the month as a decimal number [1,31] in a two-digit field with |
11355 leading space character fill.

11356 %h A synonym for %b.

11357 %H Hour (24-hour clock) as a decimal number [00,23]. |

11358 %I Hour (12-hour clock) as a decimal number [01,12]. |

%j	Day of the year as a decimal number [001,366].	
%m	Month as a decimal number [01,12].	
%M	Minute as a decimal number [00,59].	
%n	A <newline>.	
%p	Locale's equivalent of either AM or PM.	
%r	12-hour clock time [01,12] using the AM/PM notation; in the POSIX locale, this shall be equivalent to %I:%M:%S %p.	
%S	Seconds as a decimal number [00,60].	
%t	A <tab>.	
%T	24-hour clock time [00,23] in the format *HH*:*MM*:*SS*.	
%u	Weekday as a decimal number [1,7] (1=Monday).	
%U	Week of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday shall be considered to be in week 0.	
%V	Week of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing January 1 has four or more days in the new year, then it shall be considered week 1; otherwise, it shall be the last week of the previous year, and the next week shall be week 1.	
%w	Weekday as a decimal number [0,6] (0=Sunday).	
%W	Week of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday shall be considered to be in week 0.	
%x	Locale's appropriate date representation.	
%X	Locale's appropriate time representation.	
%y	Year within century [00,99].	
%Y	Year with century as a decimal number.	
%Z	Timezone name, or no characters if no timezone is determinable.	
%%	A percent sign character.	

See the Base Definitions volume of IEEE Std 1003.1-200x, Section 7.3.5, LC_TIME for the conversion specifier values in the POSIX locale.

Modified Conversion Specifications

Some conversion specifiers can be modified by the E and O modifier characters to indicate a different format or specification as specified in the *LC_TIME* locale description (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 7.3.5, LC_TIME). If the corresponding keyword (see **era**, **era_year**, **era_d_fmt**, and **alt_digits** in the Base Definitions volume of IEEE Std 1003.1-200x, Section 7.3.5, LC_TIME) is not specified or not supported for the current locale, the unmodified conversion specifier value shall be used.

%Ec	Locale's alternative appropriate date and time representation.	

%EC The name of the base year (period) in the locale's alternative representation.

%Ex Locale's alternative date representation.

%EX Locale's alternative time representation.

%Ey Offset from %EC (year only) in the locale's alternative representation.

%EY Full alternative year representation.

%Od Day of month using the locale's alternative numeric symbols.

%Oe Day of month using the locale's alternative numeric symbols.

%OH Hour (24-hour clock) using the locale's alternative numeric symbols.

%OI Hour (12-hour clock) using the locale's alternative numeric symbols.

%Om Month using the locale's alternative numeric symbols.

%OM Minutes using the locale's alternative numeric symbols.

%OS Seconds using the locale's alternative numeric symbols.

%Ou Weekday as a number in the locale's alternative representation (Monday = 1).

%OU Week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.

%OV Week number of the year (Monday as the first day of the week, rules corresponding to %V), using the locale's alternative numeric symbols.

%Ow Weekday as a number in the locale's alternative representation (Sunday = 0).

%OW Week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.

%Oy Year (offset from %C) in alternative representation.

XSI *mmddhhmm***[[***cc***]***yy* **]**

Attempt to set the system date and time from the value given in the operand. This is only possible if the user has appropriate privileges and the system permits the setting of the system date and time. The first *mm* is the month (number); *dd* is the day (number); *hh* is the hour (number, 24-hour system); the second *mm* is the minute (number); *cc* is the century and is the first two digits of the year (this is optional); *yy* is the last two digits of the year and is optional. If century is not specified, then values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive. The current year is the default if *yy* is omitted.

Note: It is expected that in a future version of IEEE Std 1003.1-200x the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *date*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

LC_TIME Determine the format and contents of date and time strings written by *date*.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

TZ Determine the timezone in which the time and date are written, unless the **–u** option is specified. If the *TZ* variable is unset or null and the **–u** is not specified, an unspecified system default timezone is used.

ASYNCHRONOUS EVENTS

Default.

STDOUT

When no formatting operand is specified, the output in the POSIX locale shall be equivalent to specifying:

```
date "+%a %b %e %H:%M:%S %Z %Y"
```

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 The date was written successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Conversion specifiers are of unspecified format when not in the POSIX locale. Some of them can contain <newline>s in some locales, so it may be difficult to use the format shown in standard output for parsing the output of *date* in those locales.

The range of values for %S extends from 0 to 60 seconds to accommodate the occasional leap second.

Although certain of the conversion specifiers in the POSIX locale (such as the name of the month) are shown with initial capital letters, this need not be the case in other locales. Programs using these fields may need to adjust the capitalization if the output is going to be used at the beginning of a sentence.

The date string formatting capabilities are intended for use in Gregorian-style calendars, possibly with a different starting year (or years). The %x and %c conversion specifications, however, are intended for local representation; these may be based on a different, non-Gregorian calendar.

The %C conversion specification was introduced to allow a fallback for the %EC (alternative year format base year); it can be viewed as the base of the current subdivision in the Gregorian calendar. The century number is calculated as the year divided by 100 and truncated to an integer; it should not be confused with the use of ordinal numbers for centuries (for example, ''twenty-first century''.) Both the %Ey and %y can then be viewed as the offset from %EC and %C, respectively.

The E and O modifiers modify the traditional conversion specifiers, so that they can always be used, even if the implementation (or the current locale) does not support the modifier.

The E modifier supports alternative date formats, such as the Japanese Emperor's Era, as long as these are based on the Gregorian calendar system. Extending the E modifiers to other date elements may provide an implementation-defined extension capable of supporting other calendar systems, especially in combination with the O modifier.

The O modifier supports time and date formats using the locale's alternative numerical symbols, such as Kanji or Hindi digits or ordinal number representation.

Non-European locales, whether they use Latin digits in computational items or not, often have local forms of the digits for use in date formats. This is not totally unknown even in Europe; a variant of dates uses Roman numerals for the months: the third day of September 1991 would be written as 3.IX.1991. In Japan, Kanji digits are regularly used for dates; in Arabic-speaking countries, Hindi digits are used. The %d, %e, %H, %I, %m, %S, %U, %w, %W, and %y conversion specifications always return the date and time field in Latin digits (that is, 0 to 9). The %O modifier was introduced to support the use for display purposes of non-Latin digits. In the *LC_TIME* category in *localedef*, the optional **alt_digits** keyword is intended for this purpose. As an example, assume the following (partial) *localedef* source:

```
alt_digits  "";"I";"II";"III";"IV";"V";"VI";"VII";"VIII" \
            "IX";"X";"XI";"XII"
d_fmt       "%e.%Om.%Y"
```

With the above date, the command:

```
date "+%x"
```

would yield 3.IX.1991. With the same **d_fmt**, but without the **alt_digits**, the command would yield 3.9.1991.

EXAMPLES

1. The following are input/output examples of *date* used at arbitrary times in the POSIX locale:

```
$ date
Tue Jun 26 09:58:10 PDT 1990

$ date "+DATE: %m/%d/%y%nTIME: %H:%M:%S"
DATE: 11/02/91
TIME: 13:36:16

$ date "+TIME: %r"
TIME: 01:36:32 PM
```

2. Examples for Denmark, where the default date and time format is %a %d %b %Y %T %Z:

```
$ LANG=da_DK.iso_8859–1 date
ons 02 okt 1991 15:03:32 CET

$ LANG=da_DK.iso_8859–1 \
    date "+DATO: %A den %e. %B %Y%nKLOKKEN: %H:%M:%S"
DATO: onsdag den 2. oktober 1991
KLOKKEN: 15:03:56
```

3. Examples for Germany, where the default date and time format is %a %d.%h.%Y, %T %Z:

```
$ LANG=De_DE.88591 date
Mi 02.Okt.1991, 15:01:21 MEZ

$ LANG=De_DE.88591 date "+DATUM: %A, %d. %B %Y%nZEIT: %H:%M:%S"
DATUM: Mittwoch, 02. Oktober 1991
ZEIT: 15:02:02
```

4. Examples for France, where the default date and time format is %a %d %h %Y %Z %T:

```
$ LANG=Fr_FR.88591 date
Mer 02 oct 1991 MET 15:03:32

$ LANG=Fr_FR.88591 date "+JOUR: %A %d %B %Y%nHEURE: %H:%M:%S"
JOUR: Mercredi 02 octobre 1991
HEURE: 15:03:56
```

RATIONALE

Some of the new options for formatting are from the ISO C standard. The **–u** option was introduced to allow portable access to Coordinated Universal Time (UTC). The string "GMT0" is allowed as an equivalent *TZ* value to be compatible with all of the systems using the BSD implementation, where this option originated.

The %e format conversion specifications (adopted from System V) was added because the ISO C standard conversion specifications did not provide any way to produce the historical default *date* output during the first nine days of any month.

There are two varieties of day and week numbering supported (in addition to any others created with the locale-dependent %E and %O modifier characters):

- The historical variety in which Sunday is the first day of the week and the weekdays preceding the first Sunday of the year are considered week 0. These are represented by %w and %U. A variant of this is %W, using Monday as the first day of the week, but still referring to week 0. This view of the calendar was retained because so many historical applications depend on it and the ISO C standard *strftime*() function, on which many *date*

implementations are based, was defined in this way.

- The international standard, based on the ISO 8601:2000 standard where Monday is the first weekday and the algorithm for the first week number is more complex: If the week (Monday to Sunday) containing January 1 has four or more days in the new year, then it is week 1; otherwise, it is week 53 of the previous year, and the next week is week 1. These are represented by the new conversion specifications %u and %V, added as a result of international comments.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *printf*( ), *strftime*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

Changes are made for Year 2000 alignment.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The setting of system date and time is described, including how to interpret two-digit year values if a century is not given.
- The %EX modified conversion specification is added.

The Open Group Corrigendum U048/2 is applied, correcting the examples.

The DESCRIPTION is updated to refer to conversion specifications, instead of field descriptors for consistency with the *LC_TIME* category.

A clarification is made such that the current year is the default if the *yy* argument is omitted when setting the system date and time.

NAME

dd — convert and copy a file

SYNOPSIS

```
dd [operand ...]
```

DESCRIPTION

The *dd* utility shall copy the specified input file to the specified output file with possible conversions using specific input and output block sizes. It shall read the input one block at a time, using the specified input block size; it shall then process the block of data actually returned, which could be smaller than the requested block size. It shall apply any conversions that have been specified and write the resulting data to the output in blocks of the specified output block size. If the **bs**=*expr* operand is specified and no conversions other than **sync**, **noerror**, or **notrunc** are requested, the data returned from each input block shall be written as a separate output block; if the read returns less than a full block and the **sync** conversion is not specified, the resulting output block shall be the same size as the input block. If the **bs**=*expr* operand is not specified, or a conversion other than **sync**, **noerror**, or **notrunc** is requested, the input shall be processed and collected into full-sized output blocks until the end of the input is reached.

The processing order shall be as follows:

1. An input block is read.
2. If the input block is shorter than the specified input block size and the **sync** conversion is specified, null bytes shall be appended to the input data up to the specified size. (If either **block** or **unblock** is also specified, <space>s shall be appended instead of null bytes.) The remaining conversions and output shall include the pad characters as if they had been read from the input.
3. If the **bs**=*expr* operand is specified and no conversion other than **sync** or **noerror** is requested, the resulting data shall be written to the output as a single block, and the remaining steps are omitted.
4. If the **swab** conversion is specified, each pair of input data bytes shall be swapped. If there is an odd number of bytes in the input block, the last byte in the input record shall not be swapped.
5. Any remaining conversions (**block**, **unblock**, **lcase**, and **ucase**) shall be performed. These conversions shall operate on the input data independently of the input blocking; an input or output fixed-length record may span block boundaries.
6. The data resulting from input or conversion or both shall be aggregated into output blocks of the specified size. After the end of input is reached, any remaining output shall be written as a block without padding if **conv**=**sync** is not specified; thus, the final output block may be shorter than the output block size.

OPTIONS

None.

OPERANDS

All of the operands shall be processed before any input is read. The following operands shall be supported:

if=*file* Specify the input pathname; the default is standard input.

of=*file* Specify the output pathname; the default is standard output. If the **seek**=*expr* conversion is not also specified, the output file shall be truncated before the copy begins if an explicit **of**=*file* operand is specified, unless **conv**=**notrunc** is specified.

If **seek**=*expr* is specified, but **conv**=**notrunc** is not, the effect of the copy shall be to preserve the blocks in the output file over which *dd* seeks, but no other portion of the output file shall be preserved. (If the size of the seek plus the size of the input file is less than the previous size of the output file, the output file shall be shortened by the copy.)

ibs=*expr* Specify the input block size, in bytes, by *expr* (default is 512).

obs=*expr* Specify the output block size, in bytes, by *expr* (default is 512).

bs=*expr* Set both input and output block sizes to *expr* bytes, superseding **ibs**= and **obs**=. If no conversion other than **sync**, **noerror**, and **notrunc** is specified, each input block shall be copied to the output as a single block without aggregating short blocks.

cbs=*expr* Specify the conversion block size for **block** and **unblock** in bytes by *expr* (default is zero). If **cbs**= is omitted or given a value of zero, using **block** or **unblock** produces unspecified results.

XSI The application shall ensure that this operand is also specified if the **conv**= operand is specified with a value of **ascii**, **ebcdic**, or **ibm**. For a **conv**= operand with an **ascii** value, the input is handled as described for the **unblock** value, except that characters are converted to ASCII before any trailing <space>s are deleted. For **conv**= operands with **ebcdic** or **ibm** values, the input is handled as described for the **block** value except that the characters are converted to EBCDIC or IBM EBCDIC, respectively, after any trailing <space>s are added.

skip=*n* Skip *n* input blocks (using the specified input block size) before starting to copy. On seekable files, the implementation shall read the blocks or seek past them; on non-seekable files, the blocks shall be read and the data shall be discarded.

seek=*n* Skip *n* blocks (using the specified output block size) from beginning of the output file before copying. On non-seekable files, existing blocks shall be read and space from the current end-of-file to the specified offset, if any, filled with null bytes; on seekable files, the implementation shall seek to the specified offset or read the blocks as described for non-seekable files.

count=*n* Copy only *n* input blocks.

conv=*value***[**,*value* ...**]**
Where *value*s are comma-separated symbols from the following list:

XSI **ascii** Convert EBCDIC to ASCII; see Table 4-6 (on page 2506).

XSI **ebcdic** Convert ASCII to EBCDIC; see Table 4-6 (on page 2506).

XSI **ibm** Convert ASCII to a different EBCDIC set; see Table 4-7 (on page 2507).

The **ascii**, **ebcdic**, and **ibm** values are mutually-exclusive.

block Treat the input as a sequence of <newline>-terminated or end-of-file-terminated variable-length records independent of the input block boundaries. Each record shall be converted to a record with a fixed length specified by the conversion block size. Any <newline> shall be removed from the input line; <space>s shall be appended to lines that are shorter than their conversion block size to fill the block. Lines that are longer than the conversion block size shall be truncated to the largest number of characters that fit into that size; the number of truncated lines shall be reported (see the STDERR

section).

The **block** and **unblock** values are mutually-exclusive.

unblock Convert fixed-length records to variable length. Read a number of bytes equal to the conversion block size (or the number of bytes remaining in the input, if less than the conversion block size), delete all trailing <space>s, and append a <newline>.

lcase Map uppercase characters specified by the *LC_CTYPE* keyword **tolower** to the corresponding lowercase character. Characters for which no mapping is specified shall not be modified by this conversion.

The **lcase** and **ucase** symbols are mutually-exclusive.

ucase Map lowercase characters specified by the *LC_CTYPE* keyword **toupper** to the corresponding uppercase character. Characters for which no mapping is specified shall not be modified by this conversion.

swab Swap every pair of input bytes.

noerror Do not stop processing on an input error. When an input error occurs, a diagnostic message shall be written on standard error, followed by the current input and output block counts in the same format as used at completion (see the STDERR section). If the **sync** conversion is specified, the missing input shall be replaced with null bytes and processed normally; otherwise, the input block shall be omitted from the output.

notrunc Do not truncate the output file. Preserve blocks in the output file not explicitly written by this invocation of the *dd* utility. (See also the preceding **of**=*file* operand.)

sync Pad every input block to the size of the **ibs**= buffer, appending null bytes. (If either **block** or **unblock** is also specified, append <space>s, rather than null bytes.)

The behavior is unspecified if operands other than **conv**= are specified more than once.

For the **bs**=, **cbs**=, **ibs**=, and **obs**= operands, the application shall supply an expression specifying a size in bytes. The expression, *expr*, can be:

1. A positive decimal number
2. A positive decimal number followed by *k*, specifying multiplication by 1 024
3. A positive decimal number followed by *b*, specifying multiplication by 512
4. Two or more positive decimal numbers (with or without *k* or *b*) separated by *x*, specifying the product of the indicated values

All of the operands are processed before any input is read.

XSI The following two tables display the octal number character values used for the **ascii** and **ebcdic** conversions (first table) and for the **ibm** conversion (second table). In both tables, the ASCII values are the row and column headers and the EBCDIC values are found at their intersections. For example, ASCII 0012 (LF) is the second row, third column, yielding 0045 in EBCDIC. The inverted tables (for EBCDIC to ASCII conversion) are not shown, but are in one-to-one correspondence with these tables. The differences between the two tables are highlighted by

small boxes drawn around five entries.

Table 4-6 ASCII to EBCDIC Conversion

	0	1	2	3	4	5	6	7
0000	0000 NUL	0001 SOH	0002 STX	0003 ETX	0067 EOT	0055 ENQ	0056 ACK	0057 BEL
	0026 BS	0005 HT	0045 LF	0013 VT	0014 FF	0015 CR	0016 SO	0017 SI
0010	0020 DLE	0021 DC1	0022 DC2	0023 DC3	0074 DC4	0075 NAK	0062 SYN	0046 ETB
	0030 CAN	0031 EM	0077 SUB	0047 ESC	0034 IFS	0035 IGS	0036 IRS	0037 ITB
0020	0100 Sp	0132 !	0177 "	0173 #	0133 $	0154 %	0120 &	0175 '
	0115 (	0135)	0134 *	0116 +	0153 ,	0140 -	0113 .	0141 /
0030	0360 0	0361 1	0362 2	0363 3	0364 4	0365 5	0366 6	0367 7
	0370 8	0371 9	0172 :	0136 ;	0114 <	0176 =	0156 >	0157 ?
0040	0174 @	0301 A	0302 B	0303 C	0304 D	0305 E	0306 F	0307 G
	0310 H	0311 I	0321 J	0322 K	0323 L	0324 M	0325 N	0326 O
0050	0327 P	0330 Q	0331 R	0342 S	0343 T	0344 U	0345 V	0346 W
	0347 X	0350 Y	0351 Z	0255 [	0340 \	0275]	0232	0155 _
0060	0171 `	0201 a	0202 b	0203 c	0204 d	0205 e	0206 f	0207 g
	0210 h	0211 i	0221 j	0222 k	0223 l	0224 m	0225 n	0226 o
0070	0227 p	0230 q	0231 r	0242 s	0243 t	0244 u	0245 v	0246 w
	0247 x	0250 y	0251 z	0300 {	0117 |	0320 }	0137 ¬	0007 DEL
0100	0040 DS	0041 SOS	0042 FS	0043 WUS	0044 BYP	0025 NL	0006 RNL	0027 POC
	0050 SA	0051 SFE	0052 SM	0053 CSP	0054 MFA	0011 SPS	0012 RPT	0033 CU1
0110	0060	0061	0032 UBS	0063 IR	0064 PP	0065 TRN	0066 NBS	0010 GE
	0070 SBS	0071 IT	0072 RFF	0073 CU3	0004 SEL	0024 RES	0076	0341
0120	0101	0102	0103	0104	0105	0106	0107	0110
	0111	0121	0122	0123	0124	0125	0126	0127
0130	0130	0131	0142	0143	0144	0145	0146	0147
	0150	0151	0160	0161	0162	0163	0164	0165
0140	0166	0167	0170	0200	0212	0213	0214	0215
	0216	0217	0220	0152 ¦	0233	0234	0235	0236
0150	0237	0240	0252	0253	0254	0112 ¢	0256	0257
	0260	0261	0262	0263	0264	0265	0266	0267
0160	0270	0271	0272	0273	0274	0241	0276	0277
	0312	0313	0314 ⌐	0315	0316 ¥	0317	0332	0333
0170	0334	0335	0336	0337	0352	0353	0354 ⊣	0355
	0356	0357	0372 ⊢	0373	0374	0375	0376	0377 EO

11728 **Table 4-7** ASCII to IBM EBCDIC Conversion

11729

	0	1	2	3	4	5	6	7
0000	0000 NUL	0001 SOH	0002 STX	0003 ETX	0067 EOT	0055 ENQ	0056 ACK	0057 BEL
0010	0026 BS	0005 HT	0045 LF	0013 VT	0014 FF	0015 CR	0016 SO	0017 SI
0020	0020 DLE	0021 DC1	0022 DC2	0023 DC3	0074 DC4	0075 NAK	0062 SYN	0046 ETB
0030	0030 CAN	0031 EM	0077 SUB	0047 ESC	0034 IFS	0035 IGS	0036 IRS	0037 ITB
0040	0100 Sp	0132 !	0177 "	0173 #	0133 $	0154 %	0120 &	0175 '
0050	0115 (	0135)	0134 *	0116 +	0153 ,	0140 -	0113 .	0141 /
0060	0360 0	0361 1	0362 2	0363 3	0364 4	0365 5	0366 6	0367 7
0070	0370 8	0371 9	0172 :	0136 ;	0114 <	0176 =	0156 >	0157 ?
0100	0174 @	0301 A	0302 B	0303 C	0304 D	0305 E	0306 F	0307 G
0110	0310 H	0311 I	0321 J	0322 K	0323 L	0324 M	0325 N	0326 O
0120	0327 P	0330 Q	0331 R	0342 S	0343 T	0344 U	0345 V	0346 W
0130	0347 X	0350 Y	0351 Z	0255 [	0340 \	0275]	0137 ¬	0155 _
0140	0171 `	0201 a	0202 b	0203 c	0204 d	0205 e	0206 f	0207 g
0150	0210 h	0211 i	0221 j	0222 k	0223]	0224 m	0225 n	0226 o
0160	0227 p	0230 q	0231 r	0242 s	0243 t	0244 u	0245 v	0246 w
0170	0247 x	0250 y	0251 z	0300 {	0117 |	0320 }	0241 ~	0007 DEL
0200	0040 DS	0041 SOS	0042 FS	0043 WUS	0044 BYP	0025 NL	0006 RNL	0027 POC
0210	0050 SA	0051 SFE	0052 SM	0053 CSP	0054 MFA	0011 SPS	0012 RPT	0033 CU1
0220	0060	0061	0032 UBS	0063 IR	0064 PP	0065 TRN	0066 NBS	0010 GE
0230	0070 SBS	0071 IT	0072 RFF	0073 CU3	0004 SEL	0024 RES	0076	0341
0240	0101	0102	0103	0104	0105	0106	0107	0110
0250	0111	0121	0122	0123	0124	0125	0126	0127
0260	0130	0131	0142	0143	0144	0145	0146	0147
0270	0150	0151	0160	0161	0162	0163	0164	0165
0300	0166	0167	0170	0200	0212	0213	0214	0215
0310	0216	0217	0220	0232	0233	0234	0235	0236
0320	0237	0240	0252	0253	0254	0255 [	0256	0257
0330	0260	0261	0262	0263	0264	0265	0266	0267
0340	0270	0271	0272	0273	0274	0275]	0276	0277
0350	0312	0313	0314 ⌐	0315	0316 ¥	0317	0332	0333
0360	0334	0335	0336	0337	0352	0353	0354 ⊣	0355
0370	0356	0357	0372 |	0373	0374	0375	0376	0377 EO

STDIN

If no **if**= operand is specified, the standard input shall be used. See the INPUT FILES section.

INPUT FILES

The input file can be any file type.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *dd*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the classification of characters as uppercase or lowercase, and the mapping of characters from one case to the other.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

For SIGINT, the *dd* utility shall interrupt its current processing, write status information to standard error, and exit as though terminated by SIGINT. It shall take the standard action for all other signals; see the ASYNCHRONOUS EVENTS section in Section 1.11 (on page 2221).

STDOUT

If no **of**= operand is specified, the standard output shall be used. The nature of the output depends on the operands selected.

STDERR

On completion, *dd* shall write the number of input and output blocks to standard error. In the POSIX locale the following formats shall be used:

```
"%u+%u records in\n", <number of whole input blocks>,
    <number of partial input blocks>

"%u+%u records out\n", <number of whole output blocks>,
    <number of partial output blocks>
```

A partial input block is one for which *read*( ) returned less than the input block size. A partial output block is one that was written with fewer bytes than specified by the output block size.

In addition, when there is at least one truncated block, the number of truncated blocks shall be written to standard error. In the POSIX locale, the format shall be:

```
"%u truncated %s\n", <number of truncated blocks>, "record" (if
    <number of truncated blocks> is one) "records" (otherwise)
```

Diagnostic messages may also be written to standard error.

11772 **OUTPUT FILES**

11773 If the **of**= operand is used, the output shall be the same as described in the STDOUT section.

11774 **EXTENDED DESCRIPTION**

11775 None.

11776 **EXIT STATUS**

11777 The following exit values shall be returned:

11778 0 The input file was copied successfully.

11779 >0 An error occurred.

11780 **CONSEQUENCES OF ERRORS**

11781 If an input error is detected and the **noerror** conversion has not been specified, any partial
11782 output block shall be written to the output file, a diagnostic message shall be written, and the
11783 copy operation shall be discontinued. If some other error is detected, a diagnostic message shall
11784 be written and the copy operation shall be discontinued.

11785 **APPLICATION USAGE**

11786 The input and output block size can be specified to take advantage of raw physical I/O.

11787 There are many different versions of the EBCDIC codesets. The ASCII and EBCDIC conversions
11788 specified for the *dd* utility perform conversions for the version specified by the tables.

11789 **EXAMPLES**

11790 The following command:

11791 ```
dd if=/dev/rmt0h of=/dev/rmt1h
```

11792 copies from tape drive 0 to tape drive 1, using a common historical device naming convention.

11793 The following command:

11794 ```
dd ibs=10  skip=1
```

11795 strips the first 10 bytes from standard input.

11796 This example reads an EBCDIC tape blocked ten 80-byte EBCDIC card images per block into the
11797 ASCII file **x**:

11798 ```
dd if=/dev/tape of=x ibs=800 cbs=80 conv=ascii,lcase
```

11799 **RATIONALE**

11800 The OPTIONS section is listed as ‘‘None’’ because there are no options recognized by historical
11801 *dd* utilities. Certainly, many of the operands could have been designed to use the Utility Syntax
11802 Guidelines, which would have resulted in the classic hyphenated option letters. In this version
11803 of this volume of IEEE Std 1003.1-200x, *dd* retains its curious JCL-like syntax due to the large
11804 number of applications that depend on the historical implementation.

11805 A suggested implementation technique for **conv**=**noerror**,**sync** is to zero (or <space>-fill, if
11806 **block**ing or **unblock**ing) the input buffer before each read and to write the contents of the input
11807 buffer to the output even after an error. In this manner, any data transferred to the input buffer
11808 before the error was detected is preserved. Another point is that a failed read on a regular file or
11809 a disk generally does not increment the file offset, and *dd* must then seek past the block on which
11810 the error occurred; otherwise, the input error occurs repetitively. When the input is a magnetic
11811 tape, however, the tape normally has passed the block containing the error when the error is
11812 reported, and thus no seek is necessary.

11813 The default **ibs**= and **obs**= sizes are specified as 512 bytes because there are historical (largely
11814 portable) scripts that assume these values. If they were left unspecified, unusual results could
```

occur if an implementation chose an odd block size.

Historical implementations of *dd* used *creat*() when processing **of**=*file*. This makes the **seek**= operand unusable except on special files. The **conv**=**notrunc** feature was added because more recent BSD-based implementations use *open*() (without O_TRUNC) instead of *creat*(), but they fail to delete output file contents after the data copied.

The *w* multiplier (historically meaning *word*), is used in System V to mean 2 and in 4.2 BSD to mean 4. Since *word* is inherently non-portable, its use is not supported by this volume of IEEE Std 1003.1-200x.

Standard EBCDIC does not have the characters '[' and ']'. The values used in the table are taken from a common print train that does contain them. Other than those characters, the print train values are not filled in, but appear to provide some of the motivation for the historical choice of translations reflected here.

The Standard EBCDIC table provides a 1:1 translation for all 256 bytes.

The IBM EBCDIC table does not provide such a translation. The marked cells in the tables differ in such a way that:

1. EBCDIC 0112 ('¢') and 0152 (broken pipe) do not appear in the table.
2. EBCDIC 0137 ('¬') translates to/from ASCII 0236 ('^'). In the standard table, EBCDIC 0232 (no graphic) is used.
3. EBCDIC 0241 ('~') translates to/from ASCII 0176 ('~'). In the standard table, EBCDIC 0137 ('¬') is used.
4. 0255 ('[') and 0275 (']') appear twice, once in the same place as for the standard table and once in place of 0112 ('¢') and 0241 ('~').

In net result:

EBCDIC 0275 (']') displaced EBCDIC 0241 ('~') in cell 0345.

That displaced EBCDIC 0137 ('¬') in cell 0176.

That displaced EBCDIC 0232 (no graphic) in cell 0136.

That replaced EBCDIC 0152 (broken pipe) in cell 0313.

EBCDIC 0255 ('[') replaced EBCDIC 0112 ('¢').

This translation, however, reflects historical practice that (ASCII) '~' and '¬' were often mapped to each other, as were '[' and '¢'; and ']' and (EBCDIC) '~'.

The **cbs** operand is required if any of the **ascii**, **ebcdic**, or **ibm** operands are specified. For the **ascii** operand, the input is handled as described for the **unblock** operand except that characters are converted to ASCII before the trailing <space>s are deleted. For the **ebcdic** and **ibm** operands, the input is handled as described for the **block** operand except that the characters are converted to EBCDIC or IBM EBCDIC after the trailing <space>s are added.

The **block** and **unblock** keywords are from historical BSD practice.

The consistent use of the word **record** in standard error messages matches most historical practice. An earlier version of System V used **block**, but this has been updated in more recent releases.

Early proposals only allowed two numbers separated by **x** to be used in a product when specifying **bs**=, **cbs**=, **ibs**=, and **obs**= sizes. This was changed to reflect the historical practice of allowing multiple numbers in the product as provided by Version 7 and all releases of System V

and BSD.

A change to the *swab* conversion is required to match historical practice and is the result of IEEE PASC Interpretation 1003.2 #03 and #04, submitted for the ISO POSIX-2: 1993 standard.

A change to the handling of SIGINT is required to match historical practice and is the result of IEEE PASC Interpretation 1003.2 #06 submitted for the ISO POSIX-2: 1993 standard.

FUTURE DIRECTIONS

None.

SEE ALSO

sed, *tr*

CHANGE HISTORY

First released in Issue 2.

Issue 5

The second paragraph of the **cbs**= description is reworded and marked EX.

FUTURE DIRECTIONS section added.

Issue 6

Changes are made to *swab* conversion and SIGINT handling to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

IEEE PASC Interpretation 1003.2 #209 is applied, clarifying the interaction between *dd* **of**=*file* and **conv**=**notrunc**.

11877 **NAME**

11878 delta — make a delta (change) to an SCCS file (**DEVELOPMENT**)

11879 **SYNOPSIS**

11880 XSI `delta [-nps][-g list][-m mrlist][-r SID][-y[comment]] file...`

11881

11882 **DESCRIPTION**

11883 The *delta* utility shall be used to permanently introduce into the named SCCS files changes that
11884 were made to the files retrieved by *get* (called the *g-files*, or generated files).

11885 **OPTIONS**

11886 The *delta* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
11887 12.2, Utility Syntax Guidelines, except that the **-y** option has an optional option-argument. This
11888 optional option-argument shall not be presented as a separate argument.

11889 The following options shall be supported:

11890 **-r** *SID* Uniquely identify which delta is to be made to the SCCS file. The use of this option
11891 shall be necessary only if two or more outstanding *get* commands for editing (*get*
11892 **-e**) on the same SCCS file were done by the same person (login name). The SID
11893 value specified with the **-r** option can be either the SID specified on the *get*
11894 command line or the SID to be made as reported by the *get* utility; see *get* (on page
11895 2675).

11896 **-s** Suppress the report to standard output of the activity associated with each *file*.
11897 See the STDOUT section.

11898 **-n** Specify retention of the edited *g-file* (normally removed at completion of delta
11899 processing).

11900 **-g** *list* Specify a *list*, (see *get* (on page 2675) for the definition of *list*) of deltas that shall be
11901 ignored when the file is accessed at the change level (SID) created by this delta.

11902 **-m** *mrlist* Specify a modification request (MR) number that the application shall supply as
11903 the reason for creating the new delta. This shall be used if the SCCS file has the **v**
11904 flag set; see *admin* (on page 2328).

11905 If **-m** is not used and `'-'` is not specified as a file argument, and the standard
11906 input is a terminal, the prompt described in the STDOUT section shall be written
11907 to standard output before the standard input is read; if the standard input is not a
11908 terminal, no prompt shall be issued.

11909 MRs in a list shall be separated by <blank>s or escaped <newline>s. An
11910 unescaped <newline> shall terminate the MR list. The escape character is
11911 <backslash>.

11912 If the **v** flag has a value, it shall be taken to be the name of a program which
11913 validates the correctness of the MR numbers. If a non-zero exit status is returned
11914 from the MR number validation program, the *delta* utility shall terminate. (It is
11915 assumed that the MR numbers were not all valid.)

11916 **-y[***comment***]** Describe the reason for making the delta. The *comment* shall be an arbitrary group
11917 of lines that would meet the definition of a text file. Implementations shall support
11918 *comment*s from zero to 512 bytes and may support longer values. A null string
11919 (specified as either **-y**, **-y**" ", or in response to a prompt for a comment) shall be
11920 considered a valid *comment*.

If **–y** is not specified and '–' is not specified as a file argument, and the standard input is a terminal, the prompt described in the STDOUT section shall be written to standard output before the standard input is read; if the standard input is not a terminal, no prompt shall be issued. An unescaped <newline> shall terminate the comment text. The escape character is <backslash>.

The **–y** option shall be required if the *file* operand is specified as '–'.

–p Write (to standard output) the SCCS file differences before and after the delta is applied in *diff* format; see *diff* (on page 2520).

OPERANDS

The following operand shall be supported:

file A pathname of an existing SCCS file or a directory. If *file* is a directory, the *delta* utility shall behave as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with **s.**) and unreadable files shall be silently ignored.

If exactly one *file* operand appears, and it is '–', the standard input shall be read; each line of the standard input shall be taken to be the name of an SCCS file to be processed. Non-SCCS files and unreadable files shall be silently ignored.

STDIN

The standard input shall be a text file used only in the following cases:

- To read an *mrlist* or a *comment* (see the **–m** and **–y** options).
- A *file* operand shall be specified as '–'. In this case, the **–y** option must be used to specify the comment, and if the SCCS file has the **v** flag set, the **–m** option must also be used to specify the MR list.

INPUT FILES

Input files shall be text files whose data is to be included in the SCCS files. If the first character of any line of an input file is <SOH> in the POSIX locale, the results are unspecified. If this file contains more than 99 999 lines, the number of lines recorded in the header for this file shall be 99 999 for this delta.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *delta*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error, and informative messages written to standard output.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

TZ Determine the timezone in which the time and date are written in the SCCS file. If the *TZ* variable is unset or NULL, an unspecified system default timezone is used.

ASYNCHRONOUS EVENTS

If SIGINT is caught, temporary files shall be cleaned up and *delta* shall exit with a non-zero exit code. The standard action shall be taken for all other signals; see Section 1.11 (on page 2221).

STDOUT

The standard output shall be used only for the following messages in the POSIX locale:

- Prompts (see the –**m** and –**y** options) in the following formats:

  ```
  "MRs? "
  ```

  ```
  "comments? "
  ```

  The MR prompt, if written, shall always precede the comments prompt.

- A report of each *file*'s activities (unless the –**s** option is specified) in the following format:

  ```
  "%s\n%d inserted\n%d deleted\n%d unchanged\n", <New SID>,
      <number of lines inserted>, <number of lines deleted>,
      <number of lines unchanged>
  ```

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

Any SCCS files updated shall be files of an unspecified format.

EXTENDED DESCRIPTION

System Date and Time

When a delta is added to an SCCS file, the system date and time shall be recorded for the new delta. If a *get* is performed using an SCCS file with a date recorded apparently in the future, the behavior is unspecified.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Problems can arise if the system date and time have been modified (for example, put forward and then back again, or unsynchronized clocks across a network) and can also arise when different values of the *TZ* environment variable are used.

Problems of a similar nature can also arise for the operation of the *get* utility, which records the date and time in the file body.

EXAMPLES

None.

12003 **RATIONALE**

12004 None.

12005 **FUTURE DIRECTIONS**

12006 None.

12007 **SEE ALSO**

12008 *admin*, *diff*, *get*, *prs*, *rmdel*

12009 **CHANGE HISTORY**

12010 First released in Issue 2.

12011 **Issue 5**

12012 The output format description in the STDOUT section is corrected.

12013 **Issue 6**

12014 The APPLICATION USAGE section is added.

12015 The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

12016 The normative text is reworded to emphasize the term ‘‘shall’’ for implementation requirements. |

12017 The Open Group Base Resolution bwg2001-007 is applied as follows: |

12018 • The use of ’–’ as a file argument is clarified. |

12019 • The use of STDIN is added. |

12020 • The ASYNCHRONOUS EVENTS section is updated to remove the implicit requirement that |
12021 implementations re-signal themselves when catching a normally fatal signal. |

12022 • New text is added to the INPUT FILES section warning that the maximum lines recorded in |
12023 the file is 99 999. |

12024 New text is added to the EXTENDED DESCRIPTION and APPLICAION USAGE sections |
12025 regarding how the system date and time may be taken into account, and the *TZ* environment |
12026 variable is added to the ENVIRONMENT VARIABLES section as per The Open Group Base |
12027 Resolution bwg2001-007. |

12028 **NAME**

12029 df — report free disk space

12030 **SYNOPSIS**

12031 UP XSI `df [–k][–P|–t][file...]`

12032

12033 **DESCRIPTION**

12034 XSI The *df* utility shall write the amount of available space and file slots for file systems on which the
12035 invoking user has appropriate read access. File systems shall be specified by the *file* operands;
12036 when none are specified, information shall be written for all file systems. The format of the
12037 default output from *df* is unspecified, but all space figures are reported in 512-byte units, unless
12038 the **–k** option is specified. This output shall contain at least the file system names, amount of
12039 XSI available space on each of these file systems, and the number of free file slots, or *inodes*,
12040 available; when **–t** is specified, the output shall contain the total allocated space as well.

12041 **OPTIONS**

12042 The *df* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
12043 Utility Syntax Guidelines.

12044 The following options shall be supported:

12045 **–k** Use 1 024-byte units, instead of the default 512-byte units, when writing space
12046 figures.

12047 **–P** Produce output in the format described in the STDOUT section.

12048 XSI **–t** Include total allocated-space figures in the output.

12049 **OPERANDS**

12050 The following operand shall be supported:

12051 *file* A pathname of a file within the hierarchy of the desired file system. If a file other
12052 XSI than a FIFO, a regular file, a directory or a special file representing the device
12053 containing the file system (for example, **/dev/dsk/0s1**) is specified, the results are
12054 unspecified. Otherwise, *df* shall write the amount of free space in the file system
12055 containing the specified *file* operand.

12056 **STDIN**

12057 Not used.

12058 **INPUT FILES**

12059 None.

12060 **ENVIRONMENT VARIABLES**

12061 The following environment variables shall affect the execution of *df*:

12062 *LANG* Provide a default value for the internationalization variables that are unset or null.
12063 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
12064 Internationalization Variables for the precedence of internationalization variables
12065 used to determine the values of locale categories.)

12066 *LC_ALL* If set to a non-empty string value, override the values of all the other
12067 internationalization variables.

12068 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
12069 characters (for example, single-byte as opposed to multi-byte characters in
12070 arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
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"
```

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"
```

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 (1 024-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.

12113 *<file system root>*
12114 The directory below which the file system hierarchy appears.

12115 XSI The output format is unspecified when **–t** is used.

12116 **STDERR**
12117 The standard error shall be used only for diagnostic messages.

12118 **OUTPUT FILES**
12119 None.

12120 **EXTENDED DESCRIPTION**
12121 None.

12122 **EXIT STATUS**
12123 The following exit values shall be returned:

12124 0 Successful completion.

12125 >0 An error occurred.

12126 **CONSEQUENCES OF ERRORS**
12127 Default.

12128 **APPLICATION USAGE**
12129 On most systems, the ‘‘name of the file system, in an implementation-defined format’’ is the
12130 special file on which the file system is mounted.

12131 On large file systems, the calculation specified for percentage used can create huge rounding
12132 errors.

12133 **EXAMPLES**

12134 1. The following example writes portable information about the **/usr** file system:

12135 ```
df –P /usr
```

12136 2. Assuming that **/usr/src** is part of the **/usr** file system, the following produces the same
12137 output as the previous example:

12138 ```
df –P /usr/src
```

12139 **RATIONALE**
12140 The behavior of *df* with the **–P** option is the default action of the 4.2 BSD *df* utility. The uppercase
12141 **–P** was selected to avoid collision with a known industry extension using **–p**.

12142 Historical *df* implementations vary considerably in their default output. It was therefore
12143 necessary to describe the default output in a loose manner to accommodate all known historical
12144 implementations and to add a portable option (**–P**) to provide information in a portable format.

12145 The use of 512-byte units is historical practice and maintains compatibility with *ls* and other
12146 utilities in this volume of IEEE Std 1003.1-200x. This does not mandate that the file system itself
12147 be based on 512-byte blocks. The **–k** option was added as a compromise measure. It was agreed
12148 by the standard developers that 512 bytes was the best default unit because of its complete
12149 historical consistency on System V (*versus* the mixed 512/1 024-byte usage on BSD systems), and
12150 that a **–k** option to switch to 1 024-byte units was a good compromise. Users who prefer the
12151 more logical 1 024-byte quantity can easily alias *df* to *df* **–k** without breaking many historical
12152 scripts relying on the 512-byte units.

12153 It was suggested that *df* and the various related utilities be modified to access a *BLOCKSIZE*
12154 environment variable to achieve consistency and user acceptance. Since this is not historical
12155 practice on any system, it is left as a possible area for system extensions and will be re-evaluated

in a future version if it is widely implemented.

FUTURE DIRECTIONS

None.

SEE ALSO

find

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

12165 **NAME**

12166 diff — compare two files

12167 **SYNOPSIS**

12168 `diff [-c| -e| -f| -C n][-br] file1 file2`

12169 **DESCRIPTION**

12170 The *diff* utility shall compare the contents of *file1* and *file2* and write to standard output a list of
12171 changes necessary to convert *file1* into *file2*. This list should be minimal. No output shall be
12172 produced if the files are identical.

12173 **OPTIONS**

12174 The *diff* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
12175 12.2, Utility Syntax Guidelines.

12176 The following options shall be supported:

12177 **–b** Cause any amount of white space at the end of a line to be treated as a single
12178 <newline> (that is, the white-space characters preceding the <newline> are
12179 ignored) and other strings of white-space characters, not including <newline>s, to
12180 compare equal.

12181 **–c** Produce output in a form that provides three lines of context.

12182 **–C n** Produce output in a form that provides *n* lines of context (where *n* shall be
12183 interpreted as a positive decimal integer).

12184 **–e** Produce output in a form suitable as input for the *ed* utility, which can then be
12185 used to convert *file1* into *file2*.

12186 **–f** Produce output in an alternative form, similar in format to **–e**, but not intended to
12187 be suitable as input for the *ed* utility, and in the opposite order.

12188 **–r** Apply *diff* recursively to files and directories of the same name when *file1* and *file2*
12189 are both directories.

12190 **OPERANDS**

12191 The following operands shall be supported:

12192 *file1*, *file2* A pathname of a file to be compared. If either the *file1* or *file2* operand is `'-'`, the
12193 standard input shall be used in its place.

12194 If both *file1* and *file2* are directories, *diff* shall not compare block special files, character special
12195 files, or FIFO special files to any files and shall not compare regular files to directories. Further
12196 details are as specified in **Diff Directory Comparison Format** (on page 2521). The behavior of
12197 *diff* on other file types is implementation-defined when found in directories.

12198 If only one of *file1* and *file2* is a directory, *diff* shall be applied to the non-directory file and the file
12199 contained in the directory file with a filename that is the same as the last component of the non-
12200 directory file.

12201 **STDIN**

12202 The standard input shall be used only if one of the *file1* or *file2* operands references standard
12203 input. See the INPUT FILES section.

12204 **INPUT FILES**

12205 The input files may be of any type.

12206 **ENVIRONMENT VARIABLES**

12207 The following environment variables shall affect the execution of *diff*:

12208 *LANG* Provide a default value for the internationalization variables that are unset or null.
12209 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
12210 Internationalization Variables for the precedence of internationalization variables
12211 used to determine the values of locale categories.)

12212 *LC_ALL* If set to a non-empty string value, override the values of all the other
12213 internationalization variables.

12214 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
12215 characters (for example, single-byte as opposed to multi-byte characters in
12216 arguments and input files).

12217 *LC_MESSAGES*
12218 Determine the locale that should be used to affect the format and contents of
12219 diagnostic messages written to standard error and informative messages written to
12220 standard output.

12221 *LC_TIME* Determine the locale for affecting the format of file timestamps written with the
12222 **–C** and **–c** options.

12223 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

12224 *TZ* Determine the timezone used for calculating file timestamps written with the **–C**
12225 and **–c** options. If *TZ* is unset or null, an unspecified default timezone shall be
12226 used.

12227 **ASYNCHRONOUS EVENTS**

12228 Default.

12229 **STDOUT**

12230 **Diff Directory Comparison Format**

12231 If both *file1* and *file2* are directories, the following output formats shall be used.

12232 In the POSIX locale, each file that is present in only one directory shall be reported using the
12233 following format:

12234 `"Only in %s: %s\n", <directory pathname>, <filename>`

12235 In the POSIX locale, subdirectories that are common to the two directories may be reported with
12236 the following format:

12237 `"Common subdirectories: %s and %s\n", <directory1 pathname>,`
12238 `<directory2 pathname>`

12239 For each file common to the two directories if the two files are not to be compared, the following
12240 format shall be used in the POSIX locale:

12241 `"File %s is a %s while file %s is a %s\n", <directory1 pathname>,`
12242 `<file type of directory1 pathname>, <directory2 pathname>,`
12243 `<file type of directory2 pathname>`

12244 For each file common to the two directories, if the files are compared and are identical, no output
12245 shall be written. If the two files differ, the following format is written:

12246 `"diff %s %s %s\n", <diff_options>, <filename1>, <filename2>`

where <*diff_options*> are the options as specified on the command line.

All directory pathnames listed in this section shall be relative to the original command line arguments. All other names of files listed in this section shall be filenames (pathname components).

Diff Binary Output Format

In the POSIX locale, if one or both of the files being compared are not text files, an unspecified format shall be used that contains the pathnames of two files being compared and the string `"differ"`.

If both files being compared are text files, depending on the options specified, one of the following formats shall be used to write the differences.

Diff Default Output Format

The default (without **–e**, **–f**, **–c**, or **–C** options) *diff* utility output shall contain lines of these forms:

`"%da%d\n", <num1>, <num2>`

`"%da%d,%d\n", <num1>, <num2>, <num3>`

`"%dd%d\n", <num1>, <num2>`

`"%d,%dd%d\n", <num1>, <num2>, <num3>`

`"%dc%d\n", <num1>, <num2>`

`"%d,%dc%d\n", <num1>, <num2>, <num3>`

`"%dc%d,%d\n", <num1>, <num2>, <num3>`

`"%d,%dc%d,%d\n", <num1>, <num2>, <num3>, <num4>`

These lines resemble *ed* subcommands to convert *file1* into *file2*. The line numbers before the action letters shall pertain to *file1*; those after shall pertain to *file2*. Thus, by exchanging *a* for *d* and reading the line in reverse order, one can also determine how to convert *file2* into *file1*. As in *ed*, identical pairs (where *num1* = *num2*) are abbreviated as a single number.

Following each of these lines, *diff* shall write to standard output all lines affected in the first file using the format:

`"<Δ%s", <line>`

and all lines affected in the second file using the format:

`">Δ%s", <line>`

If there are lines affected in both *file1* and *file2* (as with the **c** subcommand), the changes are separated with a line consisting of three hyphens:

`"---\n"`

12280 **Diff –e Output Format**

12281 With the **–e** option, a script shall be produced that shall, when provided as input to *ed*, along
12282 with an appended **w** (write) command, convert *file1* into *file2*. Only the **a** (append), **c** (change), **d**
12283 (delete), **i** (insert), and **s** (substitute) commands of *ed* shall be used in this script. Text lines,
12284 except those consisting of the single character period (`'.'`), shall be output as they appear in the
12285 file.

12286 **Diff –f Output Format**

12287 With the **–f** option, an alternative format of script shall be produced. It is similar to that
12288 produced by **–e**, with the following differences:

12289 1. It is expressed in reverse sequence; the output of **–e** orders changes from the end of the file
12290 to the beginning; the **–f** from beginning to end.

12291 2. The command form <*lines*> <*command-letter*> used by **–e** is reversed. For example,
12292 10*c* with **–e** would be *c*10 with **–f**.

12293 3. The form used for ranges of line numbers is <space>-separated, rather than comma-
12294 separated.

12295 **Diff –c or –C Output Format**

12296 With the **–c** or **–C** option, the output format shall consist of affected lines along with
12297 surrounding lines of context. The affected lines shall show which ones need to be deleted or
12298 changed in *file1*, and those added from *file2*. With the **–c** option, three lines of context, if
12299 available, shall be written before and after the affected lines. With the **–C** option, the user can
12300 specify how many lines of context are written. The exact format follows.

12301 The name and last modification time of each file shall be output in the following format:

12302 `"*** %s %s\n", file1, <file1 timestamp>`
12303 `"––– %s %s\n", file2, <file2 timestamp>`

12304 Each <*file*> field shall be the pathname of the corresponding file being compared. The pathname
12305 written for standard input is unspecified.

12306 In the POSIX locale, each <*timestamp*> field shall be equivalent to the output from the following
12307 command:

12308 `date "+%a %b %e %T %Y"`

12309 without the trailing <newline>, executed at the time of last modification of the corresponding
12310 file (or the current time, if the file is standard input).

12311 Then, the following output formats shall be applied for every set of changes.

12312 First, a line shall be written in the following format:

12313 `"***************\n"`

12314 Next, the range of lines in *file1* shall be written in the following format:

12315 `"*** %d,%d ****\n", <beginning line number>, <ending line number>`

12316 Next, the affected lines along with lines of context (unaffected lines) shall be written. Unaffected
12317 lines shall be written in the following format:

12318 `"ΔΔ%s", <unaffected_line>`

Deleted lines shall be written as:

"−Δ%s", <*deleted_line*>

Changed lines shall be written as:

"!Δ%s", <*changed_line*>

Next, the range of lines in *file2* shall be written in the following format:

"−−− %d,%d −−−−\n", <*beginning line number*>, <*ending line number*>

Then, lines of context and changed lines shall be written as described in the previous formats. Lines added from *file2* shall be written in the following format:

"+Δ%s", <*added_line*>

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 No differences were found.

1 Differences were found.

>1 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

If lines at the end of a file are changed and other lines are added, *diff* output may show this as a delete and add, as a change, or as a change and add; *diff* is not expected to know which happened and users should not care about the difference in output as long as it clearly shows the differences between the files.

EXAMPLES

If **dir1** is a directory containing a directory named **x**, **dir2** is a directory containing a directory named **x**, **dir1/x** and **dir2/x** both contain files named **date.out**, and **dir2/x** contains a file named **y**, the command:

```
diff −r dir1 dir2
```

could produce output similar to:

```
Common subdirectories: dir1/x and dir2/x
Only in dir2/x: y
diff −r dir1/x/date.out dir2/x/date.out
1c1
< Mon Jul  2 13:12:16 PDT 1990
−−−
> Tue Jun 19 21:41:39 PDT 1990
```

12359 **RATIONALE**

12360 The **–h** option was omitted because it was insufficiently specified and does not add to
12361 applications portability.

12362 Historical implementations employ algorithms that do not always produce a minimum list of
12363 differences; the current language about making every effort is the best this volume of
12364 IEEE Std 1003.1-200x can do, as there is no metric that could be employed to judge the quality of
12365 implementations against any and all file contents. The statement ''This list should be minimal''
12366 clearly implies that implementations are not expected to provide the following output when
12367 comparing two 100-line files that differ in only one character on a single line:

12368 ```
1,100c1,100
12369 all 100 lines from file1 preceded with "< "
12370 ---
12371 all 100 lines from file2 preceded with "> "
```

12372 The ''Only in'' messages required when the **–r** option is specified are not used by most historical
12373 implementations if the **–e** option is also specified. It is required here because it provides useful
12374 information that must be provided to update a target directory hierarchy to match a source
12375 hierarchy. The ''Common subdirectories'' messages are written by System V and 4.3 BSD when
12376 the **–r** option is specified. They are allowed here but are not required because they are reporting
12377 on something that is the same, not reporting a difference, and are not needed to update a target
12378 hierarchy.

12379 The **–c** option, which writes output in a format using lines of context, has been included. The
12380 format is useful for a variety of reasons, among them being much improved readability and the
12381 ability to understand difference changes when the target file has line numbers that differ from
12382 another similar, but slightly different, copy. The *patch* utility is most valuable when working
12383 with difference listings using the context format. The BSD version of **–c** takes an optional
12384 argument specifying the amount of context. Rather than overloading **–c** and breaking the Utility
12385 Syntax Guidelines for *diff*, the standard developers decided to add a separate option for
12386 specifying a context diff with a specified amount of context (**–C**). Also, the format for context
12387 *diff*s was extended slightly in 4.3 BSD to allow multiple changes that are within context lines
12388 from each other to be merged together. The output format contains an additional four asterisks
12389 after the range of affected lines in the first filename. This was to provide a flag for old programs
12390 (like old versions of *patch*) that only understand the old context format. The version of context
12391 described here does not require that multiple changes within context lines be merged, but it does
12392 not prohibit it either. The extension is upward-compatible, so any vendors that wish to retain the
12393 old version of *diff* can do so by adding the extra four asterisks (that is, utilities that currently use
12394 *diff* and understand the new merged format will also understand the old unmerged format, but
12395 not *vice versa*).

12396 The substitute command was added as an additional format for the **–e** option. This was added to
12397 provide implementations a way to fix the classic ''dot alone on a line'' bug present in many
12398 versions of *diff*. Since many implementations have fixed this bug, the standard developers
12399 decided not to standardize broken behavior, but rather to provide the necessary tool for fixing
12400 the bug. One way to fix this bug is to output two periods whenever a lone period is needed, then
12401 terminate the append command with a period, and then use the substitute command to convert
12402 the two periods into one period.

12403 The BSD-derived **–r** option was added to provide a mechanism for using *diff* to compare two file
12404 system trees. This behavior is useful, is standard practice on all BSD-derived systems, and is not
12405 easily reproducible with the *find* utility.

12406 The requirement that *diff* not compare files in some circumstances, even though they have the
12407 same name, is based on the actual output of historical implementations. The message specified
```

here is already in use when a directory is being compared to a non-directory. It is extended here to preclude the problems arising from running into FIFOs and other files that would cause *diff* to hang waiting for input with no indication to the user that *diff* was hung. In most common usage, *diff* **–r** should indicate differences in the file hierarchies, not the difference of contents of devices pointed to by the hierarchies.

Many early implementations of *diff* require seekable files. Since the System Interfaces volume of IEEE Std 1003.1-200x supports named pipes, the standard developers decided that such a restriction was unreasonable. Note also that the allowed filename – almost always refers to a pipe.

No directory search order is specified for *diff*. The historical ordering is, in fact, not optimal, in that it prints out all of the differences at the current level, including the statements about all common subdirectories before recursing into those subdirectories.

The message:

`"diff %s %s %s\n", <diff_options>, <filename1>, <filename2>`

does not vary by locale because it is the representation of a command, not an English sentence.

FUTURE DIRECTIONS

None.

SEE ALSO

cmp, *comm*, *ed*

CHANGE HISTORY

First released in Issue 2.

Issue 5

FUTURE DIRECTIONS section added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **–f** option is added.

The output format for **–c** or **–C** format is changed to align with changes to the IEEE P1003.2b draft standard resulting from IEEE PASC Interpretation 1003.2 #71.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

12438 **NAME**

12439 dirname — return the directory portion of pathname

12440 **SYNOPSIS**

12441 `dirname` *`string`*

12442 **DESCRIPTION**

12443 The *string* operand shall be treated as a pathname, as defined in the Base Definitions volume of
12444 IEEE Std 1003.1-200x, Section 3.266, Pathname. The string *string* shall be converted to the name
12445 of the directory containing the filename corresponding to the last pathname component in
12446 *string*, performing actions equivalent to the following steps in order:

12447 1. If *string* is //, skip steps 2 to 5.

12448 2. If *string* consists entirely of slash characters, *string* shall be set to a single slash character. In
12449 this case, skip steps 3 to 8.

12450 3. If there are any trailing slash characters in *string*, they shall be removed.

12451 4. If there are no slash characters remaining in *string*, *string* shall be set to a single period
12452 character. In this case, skip steps 5 to 8.

12453 5. If there are any trailing non-slash characters in *string*, they shall be removed.

12454 6. If the remaining *string* is //, it is implementation-defined whether steps 7 and 8 are skipped
12455 or processed.

12456 7. If there are any trailing slash characters in *string*, they shall be removed.

12457 8. If the remaining *string* is empty, *string* shall be set to a single slash character.

12458 The resulting string shall be written to standard output.

12459 **OPTIONS**

12460 None.

12461 **OPERANDS**

12462 The following operand shall be supported:

12463 *string* A string.

12464 **STDIN**

12465 Not used.

12466 **INPUT FILES**

12467 None.

12468 **ENVIRONMENT VARIABLES**

12469 The following environment variables shall affect the execution of *dirname*:

12470 *LANG* Provide a default value for the internationalization variables that are unset or null.
12471 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
12472 Internationalization Variables for the precedence of internationalization variables
12473 used to determine the values of locale categories.)

12474 *LC_ALL* If set to a non-empty string value, override the values of all the other
12475 internationalization variables.

12476 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
12477 characters (for example, single-byte as opposed to multi-byte characters in
12478 arguments).

12479 *LC_MESSAGES*
12480 Determine the locale that should be used to affect the format and contents of
12481 diagnostic messages written to standard error.

12482 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

12483 **ASYNCHRONOUS EVENTS**
12484 Default.

12485 **STDOUT**
12486 The *dirname* utility shall write a line to the standard output in the following format:

12487 `"%s\n", <resulting string>`

12488 **STDERR**
12489 The standard error shall be used only for diagnostic messages.

12490 **OUTPUT FILES**
12491 None.

12492 **EXTENDED DESCRIPTION**
12493 None.

12494 **EXIT STATUS**
12495 The following exit values shall be returned:

12496 0 Successful completion.

12497 >0 An error occurred.

12498 **CONSEQUENCES OF ERRORS**
12499 Default.

12500 **APPLICATION USAGE**
12501 The definition of *pathname* specifies implementation-defined behavior for pathnames starting
12502 with two slash characters. Therefore, applications shall not arbitrarily add slashes to the
12503 beginning of a pathname unless they can ensure that there are more or less than two or are
12504 prepared to deal with the implementation-defined consequences.

12505 **EXAMPLES**

12506

Command	Results
dirname /	/
dirname / /	/ or / /
dirname /*a*/*b*/	/*a*
dirname / /*a*/ /*b*/ /	/ /*a*
dirname	Unspecified
dirname a	. ($? = 0)
dirname ""	. ($? = 0)
dirname /*a*	/
dirname /*a*/*b*	/*a*
dirname a/*b*	*a*

12517 **RATIONALE**
12518 The *dirname* utility originated in System III. It has evolved through the System V releases to a
12519 version that matches the requirements specified in this description in System V Release 3. 4.3
12520 BSD and earlier versions did not include *dirname*.

12521 The behaviors of *basename* and *dirname* in this volume of IEEE Std 1003.1-200x have been
12522 coordinated so that when *string* is a valid pathname:

12523 `$(basename "`*`string`*`")`

12524 would be a valid filename for the file in the directory:

12525 `$(dirname "`*`string`*`")`

12526 This would not work for the versions of these utilities in early proposals due to the way
12527 processing of trailing slashes was specified. Consideration was given to leaving processing
12528 unspecified if there were trailing slashes, but this cannot be done; the Base Definitions volume of
12529 IEEE Std 1003.1-200x, Section 3.266, Pathname allows trailing slashes. The *basename* and *dirname*
12530 utilities have to specify consistent handling for all valid pathnames.

12531 **FUTURE DIRECTIONS**

12532 None.

12533 **SEE ALSO**

12534 *basename*, Section 2.5 (on page 2235)

12535 **CHANGE HISTORY**

12536 First released in Issue 2.

12537 **NAME**

12538 du — estimate file space usage

12539 **SYNOPSIS**

12540 UP

```
du [–a | –s][–kx][–H | –L][file ...]
```

12541

12542 **DESCRIPTION**

12543 By default, the *du* utility shall write to standard output the size of the file space allocated to, and
12544 the size of the file space allocated to each subdirectory of, the file hierarchy rooted in each of the
12545 specified files. By default, when a symbolic link is encountered on the command line or in the
12546 file hierarchy, *du* shall count the size of the symbolic link (rather than the file referenced by the
12547 link), and shall not follow the link to another portion of the file hierarchy. The size of the file
12548 space allocated to a file of type directory shall be defined as the sum total of space allocated to
12549 all files in the file hierarchy rooted in the directory plus the space allocated to the directory itself.

12550 When *du* cannot *stat*( ) files or *stat*( ) or read directories, it shall report an error condition and the
12551 final exit status is affected. Files with multiple links shall be counted and written for only one
12552 entry. The directory entry that is selected in the report is unspecified. By default, file sizes shall
12553 be written in 512-byte units, rounded up to the next 512-byte unit.

12554 **OPTIONS**

12555 The *du* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
12556 Utility Syntax Guidelines.

12557 The following options shall be supported:

12558 **–a** In addition to the default output, report the size of each file not of type directory in
12559 the file hierarchy rooted in the specified file. Regardless of the presence of the **–a**
12560 option, non-directories given as *file* operands shall always be listed.

12561 **–H** If a symbolic link is specified on the command line, *du* shall count the size of the
12562 file or file hierarchy referenced by the link.

12563 **–k** Write the files sizes in units of 1 024 bytes, rather than the default 512-byte units.

12564 **–L** If a symbolic link is specified on the command line or encountered during the
12565 traversal of a file hierarchy, *du* shall count the size of the file or file hierarchy
12566 referenced by the link.

12567 **–s** Instead of the default output, report only the total sum for each of the specified
12568 files.

12569 **–x** When evaluating file sizes, evaluate only those files that have the same device as
12570 the file specified by the *file* operand.

12571 Specifying more than one of the mutually-exclusive options **–H** and **–L** shall not be considered
12572 an error. The last option specified shall determine the behavior of the utility.

12573 **OPERANDS**

12574 The following operand shall be supported:

12575 *file* The pathname of a file whose size is to be written. If no *file* is specified, the current
12576 directory shall be used.

12577 **STDIN**

12578 Not used.

12579 **INPUT FILES**

12580 None.

12581 **ENVIRONMENT VARIABLES**

12582 The following environment variables shall affect the execution of *du*:

12583 *LANG* Provide a default value for the internationalization variables that are unset or null.
12584 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
12585 Internationalization Variables for the precedence of internationalization variables
12586 used to determine the values of locale categories.)

12587 *LC_ALL* If set to a non-empty string value, override the values of all the other
12588 internationalization variables.

12589 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
12590 characters (for example, single-byte as opposed to multi-byte characters in
12591 arguments).

12592 *LC_MESSAGES*
12593 Determine the locale that should be used to affect the format and contents of
12594 diagnostic messages written to standard error.

12595 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

12596 **ASYNCHRONOUS EVENTS**

12597 Default.

12598 **STDOUT**

12599 The output from *du* shall consist of the amount of the space allocated to a file and the name of
12600 the file, in the following format:

12601 `"%d %s\n", <size>, <pathname>`

12602 **STDERR**

12603 The standard error shall be used only for diagnostic messages.

12604 **OUTPUT FILES**

12605 None.

12606 **EXTENDED DESCRIPTION**

12607 None.

12608 **EXIT STATUS**

12609 The following exit values shall be returned:

12610 0 Successful completion.

12611 >0 An error occurred.

12612 **CONSEQUENCES OF ERRORS**

12613 Default.

12614 **APPLICATION USAGE**

12615 None.

12616 **EXAMPLES**

12617 None.

12618 **RATIONALE**

12619 The use of 512-byte units is historical practice and maintains compatibility with *ls* and other
12620 utilities in this volume of IEEE Std 1003.1-200x. This does not mandate that the file system itself
12621 be based on 512-byte blocks. The **–k** option was added as a compromise measure. It was agreed
12622 by the standard developers that 512 bytes was the best default unit because of its complete
12623 historical consistency on System V (*versus* the mixed 512/1 024-byte usage on BSD systems), and
12624 that a **–k** option to switch to 1 024-byte units was a good compromise. Users who prefer the
12625 1 024-byte quantity can easily alias *du* to *du* **–k** without breaking the many historical scripts
12626 relying on the 512-byte units.

12627 The **–b** option was added to an early proposal to provide a resolution to the situation where
12628 System V and BSD systems give figures for file sizes in *blocks*, which is an implementation-
12629 defined concept. (In common usage, the block size is 512 bytes for System V and 1 024 bytes for
12630 BSD systems.) However, **–b** was later deleted, since the default was eventually decided as 512-
12631 byte units.

12632 Historical file systems provided no way to obtain exact figures for the space allocation given to
12633 files. There are two known areas of inaccuracies in historical file systems: cases of *indirect blocks*
12634 being used by the file system or *sparse* files yielding incorrectly high values. An indirect block is
12635 space used by the file system in the storage of the file, but that need not be counted in the space
12636 allocated to the file. A *sparse* file is one in which an *lseek*( ) call has been made to a position
12637 beyond the end of the file and data has subsequently been written at that point. A file system
12638 need not allocate all the intervening zero-filled blocks to such a file. It is up to the
12639 implementation to define exactly how accurate its methods are.

12640 The **–a** and **–s** options were mutually-exclusive in the original version of *du*. The POSIX Shell
12641 and Utilities description is implied by the language in the SVID where **–s** is described as causing
12642 ‘‘only the grand total’’ to be reported. Some systems may produce output for **–sa**, but a Strictly
12643 Conforming POSIX Shell and Utilities Application cannot use that combination.

12644 The **–a** and **–s** options were adopted from the SVID except that the System V behavior of not
12645 listing non-directories explicitly given as operands, unless the **–a** option is specified, was
12646 considered a bug; the BSD-based behavior (report for all operands) is mandated. The default
12647 behavior of *du* in the SVID with regard to reporting the failure to read files (it produces no
12648 messages) was considered counter-intuitive, and thus it was specified that the POSIX Shell and
12649 Utilities default behavior shall be to produce such messages. These messages can be turned off
12650 with shell redirection to achieve the System V behavior.

12651 The **–x** option is historical practice on recent BSD systems. It has been adopted by this volume of
12652 IEEE Std 1003.1-200x because there was no other historical method of limiting the *du* search to a
12653 single file hierarchy. This limitation of the search is necessary to make it possible to obtain file
12654 space usage information about a file system on which other file systems are mounted, without
12655 having to resort to a lengthy *find* and *awk* script.

12656 **FUTURE DIRECTIONS**

12657 None.

SEE ALSO

ls

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The APPLICATION USAGE section is added.

This utility is reinstated, as the LEGACY marking was incorrect in Issue 5.

The obsolescent **–r** option has been removed.

The Open Group Corrigendum U025/3 is applied. The *du* utility had incorrectly been marked LEGACY.

The **–H** and **–L** options for symbolic links are added as described in the IEEE P1003.2b draft standard.

12671 **NAME**

12672 echo — write arguments to standard output

12673 **SYNOPSIS**

12674 `echo [`*`string ...`*`]`

12675 **DESCRIPTION**

12676 The *echo* utility writes its arguments to standard output, followed by a <newline>. If there are
12677 no arguments, only the <newline> is written.

12678 **OPTIONS**

12679 The *echo* utility shall not recognize the "−−" argument in the manner specified by Guideline 10
12680 of the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines;
12681 "−−" shall be recognized as a string operand.

12682 Implementations shall not support any options.

12683 **OPERANDS**

12684 The following operands shall be supported:

12685 *string* A string to be written to standard output. If any operand is **−n**, it shall be treated as
12686 a string, not an option. The following character sequences shall be recognized
12687 within any of the arguments:

12688 `\a` Write an <alert>.

12689 `\b` Write a <backspace>.

12690 `\c` Suppress the <newline> that otherwise follows the final argument in the
12691 output. All characters following the `'\c'` in the arguments shall be
12692 ignored.

12693 `\f` Write a <form-feed>.

12694 `\n` Write a <newline>.

12695 `\r` Write a <carriage-return>.

12696 `\t` Write a <tab>.

12697 `\v` Write a <vertical-tab>.

12698 `\\` Write a backslash character.

12699 `\0`*`num`* Write an 8-bit value that is the zero, one, two, or three-digit octal number
12700 *num*.

12701 **STDIN**

12702 Not used.

12703 **INPUT FILES**

12704 None.

12705 **ENVIRONMENT VARIABLES**

12706 The following environment variables shall affect the execution of *echo*:

12707 *LANG* Provide a default value for the internationalization variables that are unset or null.
12708 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
12709 Internationalization Variables for the precedence of internationalization variables
12710 used to determine the values of locale categories.)

12711 *LC_ALL* If set to a non-empty string value, override the values of all the other
12712 internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
The *echo* utility arguments shall be separated by single <space>s and a <newline> shall follow the last argument. Output transformations shall occur based on the escape sequences in the input. See the OPERANDS section.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
In the ISO/IEC 9945-2:1993 standard, it was not possible to use *echo* portably across all systems that were not XSI-conformant unless both **–n** (as the first argument) and escape sequences were omitted.

The *printf* utility can be used portably to emulate any of the traditional behaviors of the *echo* utility as follows:

- The historic System V *echo* and the current requirements in this volume of IEEE Std 1003.1-200x are equivalent to:

```
printf "%b\n" "$*"
```

- The BSD *echo* is equivalent to:

```
if [ "X$1" = "X–n" ]
then
    shift
    printf "%s" "$*"
else
    printf "%s\n" "$*"
fi
```

New applications are encouraged to use *printf* instead of *echo*.

EXAMPLES

None.

RATIONALE

The *echo* utility has not been made obsolescent because of its extremely widespread use in historical applications. Conforming applications that wish to do prompting without <newline>s or that could possibly be expecting to echo a **–n**, should use the *printf* utility derived from the Ninth Edition system.

As specified, *echo* writes its arguments in the simplest of ways. The two different historical versions of *echo* vary in fatally incompatible ways.

The BSD *echo* checks the first argument for the string **–n** which causes it to suppress the <newline> that would otherwise follow the final argument in the output.

The System V *echo* does not support any options, but allows escape sequences within its operands, as described in the OPERANDS section.

The *echo* utility does not support Utility Syntax Guideline 10 because historical applications depend on *echo* to echo *all* of its arguments, except for the **–n** option in the BSD version.

FUTURE DIRECTIONS

None.

SEE ALSO

printf

CHANGE HISTORY

First released in Issue 2.

Issue 5

In the OPTIONS section, the last sentence is changed to indicate that implementations ‘‘do not’’ support any options; in the previous issue this said ‘‘need not’’.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- A set of character sequences is defined as *string* operands.
- *LC_CTYPE* is added to the list of environment variables affecting *echo*.
- In the OPTIONS section, implementations shall not support any options.

12786 **NAME**

12787 ed — edit text

12788 **SYNOPSIS**

12789 `ed [–p` *`string`*`][–s][`*`file`*`]`

12790 **DESCRIPTION**

12791 The *ed* utility is a line-oriented text editor that uses two modes: *command mode* and *input mode*.
12792 In command mode the input characters shall be interpreted as commands, and in input mode
12793 they shall be interpreted as text. See the EXTENDED DESCRIPTION section.

12794 **OPTIONS**

12795 The *ed* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
12796 Utility Syntax Guidelines.

12797 The following options shall be supported:

12798 **–p** *string* Use *string* as the prompt string when in command mode. By default, there shall be
12799 no prompt string.

12800 **–s** Suppress the writing of byte counts by **e**, **E**, **r**, and **w** commands and of the `'!'`
12801 prompt after a !*command*.

12802 **OPERANDS**

12803 The following operand shall be supported:

12804 *file* If the *file* argument is given, *ed* shall simulate an **e** command on the file named by
12805 the pathname, *file*, before accepting commands from the standard input. If the *file*
12806 operand is `'–'`, the results are unspecified.

12807 **STDIN**

12808 The standard input shall be a text file consisting of commands, as described in the EXTENDED
12809 DESCRIPTION section.

12810 **INPUT FILES**

12811 The input files shall be text files.

12812 **ENVIRONMENT VARIABLES**

12813 The following environment variables shall affect the execution of *ed*:

12814 *HOME* Determine the pathname of the user's home directory.

12815 *LANG* Provide a default value for the internationalization variables that are unset or null.
12816 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
12817 Internationalization Variables for the precedence of internationalization variables
12818 used to determine the values of locale categories.)

12819 *LC_ALL* If set to a non-empty string value, override the values of all the other
12820 internationalization variables.

12821 *LC_COLLATE*
12822 Determine the locale for the behavior of ranges, equivalence classes, and multi-
12823 character collating elements within regular expressions.

12824 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
12825 characters (for example, single-byte as opposed to multi-byte characters in
12826 arguments and input files) and the behavior of character classes within regular
12827 expressions.

12828 *LC_MESSAGES*
12829 Determine the locale that should be used to affect the format and contents of

diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

The *ed* utility shall take the standard action for all signals (see the ASYNCHRONOUS EVENTS section in Section 1.11 (on page 2221)) with the following exceptions:

SIGINT The *ed* utility shall interrupt its current activity, write the string "?\n" to standard output, and return to command mode (see the EXTENDED DESCRIPTION section).

SIGHUP If the buffer is not empty and has changed since the last write, the *ed* utility shall attempt to write a copy of the buffer in a file. First, the file named **ed.hup** in the current directory shall be used; if that fails, the file named **ed.hup** in the directory named by the *HOME* environment variable shall be used. In any case, the *ed* utility shall exit without returning to command mode.

SIGQUIT The *ed* utility shall ignore this event.

STDOUT

Various editing commands and the prompting feature (see **–p**) write to standard output, as described in the EXTENDED DESCRIPTION section.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

The output files shall be text files whose formats are dependent on the editing commands given.

EXTENDED DESCRIPTION

The *ed* utility shall operate on a copy of the file it is editing; changes made to the copy shall have no effect on the file until a **w** (write) command is given. The copy of the text is called the *buffer*.

Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses very often can be omitted. If the **–p** option is specified, the prompt string shall be written to standard output before each command is read.

In general, only one command can appear on a line. Certain commands allow text to be input. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, no commands shall be recognized; all input is merely collected. Input mode is terminated by entering a line consisting of two characters: a period ('.') followed by a <newline>. This line is not considered part of the input text.

Regular Expressions in ed

The *ed* utility shall support basic regular expressions, as described in the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.3, Basic Regular Expressions. Since regular expressions in *ed* are always matched against single lines (excluding the terminating <newline>s), never against any larger section of text, there is no way for a regular expression to match a <newline>.

A null RE shall be equivalent to the last RE encountered.

Regular expressions are used in addresses to specify lines, and in some commands (for example, the **s** substitute command) to specify portions of a line to be substituted.

12874 **Addresses in ed**

12875 Addressing in *ed* relates to the current line. Generally, the current line is the last line affected by a
12876 command. The current line number is the address of the current line. If the edit buffer is not
12877 empty, the initial value for the current line shall be the last line in the edit buffer; otherwise, zero.

12878 Addresses shall be constructed as follows:

12879 1. The period character (`'.'`) shall address the current line.

12880 2. The dollar sign character (`'$'`) shall address the last line of the edit buffer.

12881 3. The positive decimal number *n* shall address the *n*th line of the edit buffer.

12882 4. The apostrophe-x character pair (`"'x"`) shall address the line marked with the mark name
12883 character *x*, which shall be a lowercase letter from the portable character set. It shall be an
12884 error if the character has not been set to mark a line or if the line that was marked is not
12885 currently present in the edit buffer.

12886 5. A BRE enclosed by slash characters (`'/'`) shall address the first line found by searching
12887 forwards from the line following the current line toward the end of the edit buffer and
12888 stopping at the first line for which the line excluding the terminating <newline> matches
12889 the BRE. The BRE consisting of a null BRE delimited by a pair of slash characters shall
12890 address the next line for which the line excluding the terminating <newline> matches the
12891 last BRE encountered. In addition, the second slash can be omitted at the end of a
12892 command line. Within the BRE, a backslash-slash pair (`"\/"`) shall represent a literal slash
12893 instead of the BRE delimiter. If necessary, the search shall wrap around to the beginning of
12894 the buffer and continue up to and including the current line, so that the entire buffer is
12895 searched.

12896 6. A BRE enclosed by question-mark characters (`'?'`) shall address the first line found by
12897 searching backwards from the line preceding the current line toward the beginning of the
12898 edit buffer and stopping at the first line for which the line excluding the terminating
12899 <newline> matches the BRE. The BRE consisting of a null BRE delimited by a pair of
12900 question-mark characters (`"??"`) shall address the previous line for which the line
12901 excluding the terminating <newline> matches the last BRE encountered. In addition, the
12902 second question-mark can be omitted at the end of a command line. Within the BRE, a
12903 backslash-question-mark pair (`"\?"`) shall represent a literal question mark instead of the
12904 BRE delimiter. If necessary, the search shall wrap around to the end of the buffer and
12905 continue up to and including the current line, so that the entire buffer is searched.

12906 7. A plus-sign (`'+'`) or hyphen character (`'-'`) followed by a decimal number shall address
12907 the current line plus or minus the number. A plus-sign or hyphen character not followed
12908 by a decimal number shall address the current line plus or minus 1.

12909 Addresses can be followed by zero or more address offsets, optionally <blank>-separated.
12910 Address offsets are constructed as follows:

12911 - A plus-sign or hyphen character followed by a decimal number shall add or subtract,
12912 respectively, the indicated number of lines to or from the address. A plus-sign or hyphen
12913 character not followed by a decimal number shall add or subtract 1 to or from the address.

12914 - A decimal number shall add the indicated number of lines to the address.

12915 It shall not be an error for an intermediate address value to be less than zero or greater than the
12916 last line in the edit buffer. It shall be an error for the final address value to be less than zero or
12917 greater than the last line in the edit buffer. It shall be an error if a search for a BRE fails to find a
12918 matching line.

Commands accept zero, one, or two addresses. If more than the required number of addresses are provided to a command that requires zero addresses, it shall be an error. Otherwise, if more than the required number of addresses are provided to a command, the addresses specified first shall be evaluated and then discarded until the maximum number of valid addresses remain, for the specified command.

Addresses shall be separated from each other by a comma (`','`) or semicolon character (`';'`). In the case of a semicolon separator, the current line (`'.'`) shall be set to the first address, and only then will the second address be calculated. This feature can be used to determine the starting line for forwards and backwards searches; see rules 5. and 6.

Addresses can be omitted on either side of the comma or semicolon separator, in which case the resulting address pairs shall be as follows:

Specified	Resulting
`,`	`1 , $`
`, addr`	`1 ,a ddr`
`addr ,`	`addr , addr`
`;`	`. ; $`
`; addr`	`. ; addr`
`addr ;`	`addr ; addr`

Any <blank>s included between addresses, address separators, or address offsets shall be ignored.

Commands in ed

In the following list of *ed* commands, the default addresses are shown in parentheses. The number of addresses shown in the default shall be the number expected by the command. The parentheses are not part of the address; they show that the given addresses are the default.

It is generally invalid for more than one command to appear on a line. However, any command (except **e**, **E**, **f**, **q**, **Q**, **r**, **w**, and **!**) can be suffixed by the letter **l**, **n**, or **p**; in which case, except for the **l**, **n**, and **p** commands, the command shall be executed and then the new current line shall be written as described below under the **l**, **n**, and **p** commands. When an **l**, **n**, or **p** suffix is used with an **l**, **n**, or **p** command, the command shall write to standard output as described below, but it is unspecified whether the suffix writes the current line again in the requested format or whether the suffix has no effect. For example, the **pl** command (base **p** command with an **l** suffix) shall either write just the current line or write it twice—once as specified for **p** and once as specified for **l**. Also, the **g**, **G**, **v**, and **V** commands shall take a command as a parameter.

Each address component can be preceded by zero or more <blank>s. The command letter can be preceded by zero or more <blank>s. If a suffix letter (**l**, **n**, or **p**) is given, the application shall ensure that it immediately follows the command.

The **e**, **E**, **f**, **r**, and **w** commands shall take an optional *file* parameter, separated from the command letter by one or more <blank>s.

If changes have been made in the buffer since the last **w** command that wrote the entire buffer, *ed* shall warn the user if an attempt is made to destroy the editor buffer via the **e** or **q** commands. The *ed* utility shall write the string:

`"?\n"`

(followed by an explanatory message if *help mode* has been enabled via the **H** command) to standard output and shall continue in command mode with the current line number unchanged. If the **e** or **q** command is repeated with no intervening command, it shall take effect.

If a terminal disconnect is detected:

- If the buffer is not empty and has changed since the last write, the *ed* utility shall attempt to write a copy of the buffer to a file named **ed.hup** in the current directory. If this write fails, *ed* shall attempt to write a copy of the buffer to a filename **ed.hup** in the directory named by the *HOME* environment variable. If both these attempts fail, *ed* shall exit without saving the buffer.
- The *ed* utility shall not write the file to the currently remembered pathname or return to command mode, and shall terminate with a non-zero exit status.

If an end-of-file is detected on standard input:

- If the *ed* utility is in input mode, *ed* shall terminate input mode and return to command mode. It is unspecified if any partially entered lines (that is, input text without a terminating <newline>) are discarded from the input text.
- If the *ed* utility is in command mode, it shall act as if a **q** command had been entered.

If the closing delimiter of an RE or of a replacement string (for example, `'/'`) in a **g**, **G**, **s**, **v**, or **V** command would be the last character before a <newline>, that delimiter can be omitted, in which case the addressed line shall be written. For example, the following pairs of commands are equivalent:

```
s/s1/s2     s/s1/s2/p
g/s1        g/s1/p
?s1         ?s1?
```

If an invalid command is entered, *ed* shall write the string:

```
"?\n"
```

(followed by an explanatory message if *help mode* has been enabled via the **H** command) to standard output and shall continue in command mode with the current line number unchanged.

Append Command

Synopsis:

```
(.)a
<text>
.
```

The **a** command shall read the given text and append it after the addressed line; the current line number shall become the address of the last inserted line or, if there were none, the addressed line. Address 0 shall be valid for this command; it shall cause the appended text to be placed at the beginning of the buffer.

Change Command

Synopsis:

```
(.,.)c
<text>
.
```

The **c** command shall delete the addressed lines, then accept input text that replaces these lines; the current line shall be set to the address of the last line input; or, if there were none, at the line after the last line deleted; if the lines deleted were originally at the end of the buffer, the current line number shall be set to the address of the new last line; if no lines remain in the buffer, the current line number shall be set to zero. Address 0 shall be valid for this command; it shall be interpreted as if address 1 were specified.

13006 **Delete Command**

13007 *Synopsis*: `(.,.)d`

13008 The **d** command shall delete the addressed lines from the buffer. The address of the line after the
13009 last line deleted shall become the current line number; if the lines deleted were originally at the
13010 end of the buffer, the current line number shall be set to the address of the new last line; if no
13011 lines remain in the buffer, the current line number shall be set to zero.

13012 **Edit Command**

13013 *Synopsis*: `e [`*`file`*`]`

13014 The **e** command shall delete the entire contents of the buffer and then read in the file named by
13015 the pathname *file*. The current line number shall be set to the address of the last line of the
13016 buffer. If no pathname is given, the currently remembered pathname, if any, shall be used (see
13017 the **f** command). The number of bytes read shall be written to standard output, unless the –**s**
13018 option was specified, in the following format:

13019 `"%d\n", <`*`number of bytes read`*`>`

13020 The name *file* shall be remembered for possible use as a default pathname in subsequent **e**, **E**, **r**,
13021 and **w** commands. If *file* is replaced by `'!'`, the rest of the line shall be taken to be a shell
13022 command line whose output is to be read. Such a shell command line shall not be remembered
13023 as the current *file*. All marks shall be discarded upon the completion of a successful **e** command.
13024 If the buffer has changed since the last time the entire buffer was written, the user shall be
13025 warned, as described previously.

13026 **Edit Without Checking Command**

13027 *Synopsis*: `E [`*`file`*`]`

13028 The **E** command shall possess all properties and restrictions of the **e** command except that the
13029 editor shall not check to see whether any changes have been made to the buffer since the last **w**
13030 command.

13031 **Filename Command**

13032 *Synopsis*: `f [`*`file`*`]`

13033 If *file* is given, the **f** command shall change the currently remembered pathname to *file*; whether
13034 the name is changed or not, it shall then write the (possibly new) currently remembered
13035 pathname to the standard output in the following format:

13036 `"%s\n", <`*`pathname`*`>`

13037 The current line number shall be unchanged.

13038 **Global Command**

13039 *Synopsis*: `(1,$)g/`*`RE`*`/`*`command list`*

13040 In the **g** command, the first step shall be to mark every line for which the line excluding the
13041 terminating <newline> matches the given *RE*. Then, going sequentially from the beginning of
13042 the file to the end of the file, the given *command list* shall be executed for each marked line, with
13043 the current line number set to the address of that line. Any line modified by the *command list*
13044 shall be unmarked. When the **g** command completes, the current line number shall have the
13045 value assigned by the last command in the *command list*. If there were no matching lines, the
13046 current line number shall not be changed. A single command or the first of a list of commands

shall appear on the same line as the global command. All lines of a multi-line list except the last line shall be ended with a backslash preceding the terminating <newline>; the **a**, **i**, and **c** commands and associated input are permitted. The `'.'` terminating input mode can be omitted if it would be the last line of the *command list*. An empty *command list* shall be equivalent to the **p** command. The use of the **g**, **G**, **v**, **V**, and **!** commands in the *command list* produces undefined results. Any character other than <space> or <newline> can be used instead of a slash to delimit the *RE*. Within the *RE*, the *RE* delimiter itself can be used as a literal character if it is preceded by a backslash.

Interactive Global Command

Synopsis: `(1,$)G/`*RE*`/`

In the **G** command, the first step shall be to mark every line for which the line excluding the terminating <newline> matches the given *RE*. Then, for every such line, that line shall be written, the current line number shall be set to the address of that line, and any one command (other than one of the **a**, **c**, **i**, **g**, **G**, **v**, and **V** commands) shall be read and executed. A <newline> shall act as a null command (causing no action to be taken on the current line); an `'&'` shall cause the re-execution of the most recent non-null command executed within the current invocation of **G**. Note that the commands input as part of the execution of the **G** command can address and affect any lines in the buffer. The final value of the current line number shall be the value set by the last command successfully executed. (Note that the last command successfully executed shall be the **G** command itself if a command fails or the null command is specified.) If there were no matching lines, the current line number shall not be changed. The **G** command can be terminated by a SIGINT signal. Any character other than <space> or <newline> can be used instead of a slash to delimit the *RE* and the replacement. Within the *RE*, the *RE* delimiter itself can be used as a literal character if it is preceded by a backslash.

Help Command

Synopsis: `h`

The **h** command shall write a short message to standard output that explains the reason for the most recent `'?'` notification. The current line number shall be unchanged.

Help-Mode Command

Synopsis: `H`

The **H** command shall cause *ed* to enter a mode in which help messages (see the **h** command) shall be written to standard output for all subsequent `'?'` notifications. The **H** command alternately shall turn this mode on and off; it is initially off. If the help-mode is being turned on, the **H** command also explains the previous `'?'` notification, if there was one. The current line number shall be unchanged.

Insert Command

Synopsis:
```
(.)i
<text>
.
```

The **i** command shall insert the given text before the addressed line; the current line is set to the last inserted line or, if there was none, to the addressed line. This command differs from the **a** command only in the placement of the input text. Address 0 shall be valid for this command; it shall be interpreted as if address 1 were specified.

13090 **Join Command**

13091 *Synopsis*: `(.,.+1)j`

13092 The **j** command shall join contiguous lines by removing the appropriate <newline>s. If exactly
13093 one address is given, this command shall do nothing. If lines are joined, the current line number
13094 shall be set to the address of the joined line; otherwise, the current line number shall be
13095 unchanged.

13096 **Mark Command**

13097 *Synopsis*: `(.)k`*x*

13098 The **k** command shall mark the addressed line with name *x*, which the application shall ensure is
13099 a lowercase letter from the portable character set. The address `"'x"` shall then refer to this line;
13100 the current line number shall be unchanged.

13101 **List Command**

13102 *Synopsis*: `(.,.)l`

13103 The **l** command shall write to standard output the addressed lines in a visually unambiguous
13104 form. The characters listed in the Base Definitions volume of IEEE Std 1003.1-200x, Table 5-1,
13105 Escape Sequences and Associated Actions (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\r'`, `'\t'`, `'\v'`) shall
13106 be written as the corresponding escape sequence; the `'\n'` in that table is not applicable. Non-
13107 printable characters not in the table shall be written as one three-digit octal number (with a
13108 preceding backslash character) for each byte in the character (most significant byte first). If the
13109 size of a byte on the system is greater than nine bits, the format used for non-printable characters
13110 is implementation-defined.

13111 Long lines shall be folded, with the point of folding indicated by <newline> preceded by a
13112 backslash; the length at which folding occurs is unspecified, but should be appropriate for the
13113 output device. The end of each line shall be marked with a `'$'`, and `'$'` characters within the
13114 text shall be written with a preceding backslash. An **l** command can be appended to any other
13115 command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**. The current line number shall be set to the address of
13116 the last line written.

13117 **Move Command**

13118 *Synopsis*: `(.,.)m`*address*

13119 The **m** command shall reposition the addressed lines after the line addressed by *address*.
13120 Address 0 shall be valid for *address* and cause the addressed lines to be moved to the beginning
13121 of the buffer. It shall be an error if address *address* falls within the range of moved lines. The
13122 current line number shall be set to the address of the last line moved.

13123 **Number Command**

13124 *Synopsis*: `(.,.)n`

13125 The **n** command shall write to standard output the addressed lines, preceding each line by its
13126 line number and a <tab>; the current line number shall be set to the address of the last line
13127 written. The **n** command can be appended to any command other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

13128 **Print Command**

13129 *Synopsis*: `(.,.)p`

13130 The **p** command shall write to standard output the addressed lines; the current line number shall
13131 be set to the address of the last line written. The **p** command can be appended to any command
13132 other than **e**, **E**, **f**, **q**, **Q**, **r**, **w**, or **!**.

13133 **Prompt Command**

13134 *Synopsis*: `P`

13135 The **P** command shall cause *ed* to prompt with an asterisk (`'*'`) (or *string*, if **–p** is specified) for
13136 all subsequent commands. The **P** command alternatively shall turn this mode on and off; it shall
13137 be initially on if the **–p** option is specified; otherwise, off. The current line number shall be
13138 unchanged.

13139 **Quit Command**

13140 *Synopsis*: `q`

13141 The **q** command shall cause *ed* to exit. If the buffer has changed since the last time the entire
13142 buffer was written, the user shall be warned, as described previously.

13143 **Quit Without Checking Command**

13144 *Synopsis*: `Q`

13145 The **Q** command shall cause *ed* to exit without checking whether changes have been made in the
13146 buffer since the last **w** command.

13147 **Read Command**

13148 *Synopsis*: `($)r [`*`file`*`]`

13149 The **r** command shall read in the file named by the pathname *file* and append it after the
13150 addressed line. If no *file* argument is given, the currently remembered pathname, if any, shall be
13151 used (see the **e** and **f** commands). The currently remembered pathname shall not be changed
13152 unless there is no remembered pathname. Address 0 shall be valid for **r** and shall cause the file to
13153 be read at the beginning of the buffer. If the read is successful, and **–s** was not specified, the
13154 number of bytes read shall be written to standard output in the following format:

13155 `"%d\n", <`*`number of bytes read`*`>`

13156 The current line number shall be set to the address of the last line read in. If *file* is replaced by
13157 `'!'`, the rest of the line shall be taken to be a shell command line whose output is to be read.
13158 Such a shell command line shall not be remembered as the current pathname.

13159 **Substitute Command**

13160 *Synopsis*: `(.,.)s/`*`RE/replacement/flags`*

13161 The **s** command shall search each addressed line for an occurrence of the specified RE and
13162 replace either the first or all (non-overlapped) matched strings with the *replacement*; see the
13163 following description of the **g** suffix. It is an error if the substitution fails on every addressed
13164 line. Any character other than <space> or <newline> can be used instead of a slash to delimit the
13165 *RE* and the replacement. Within the *RE*, the *RE* delimiter itself can be used as a literal character if
13166 it is preceded by a backslash. The current line shall be set to the address of the last line on which
13167 a substitution occurred.

13168 An ampersand (’&’) appearing in the *replacement* shall be replaced by the string matching the
13169 RE on the current line. The special meaning of ’&’ in this context can be suppressed by
13170 preceding it by backslash. As a more general feature, the characters ’\n’, where *n* is a digit,
13171 shall be replaced by the text matched by the corresponding back-reference expression. When the
13172 character ’%’ is the only character in the *replacement*, the *replacement* used in the most recent
13173 substitute command shall be used as the *replacement* in the current substitute command; if there
13174 was no previous substitute command, the use of ’%’ in this manner shall be an error. The ’%’
13175 shall lose its special meaning when it is in a replacement string of more than one character or is
13176 preceded by a backslash. For each backslash (’\’) encountered in scanning *replacement* from
13177 beginning to end, the following character shall lose its special meaning (if any). It is unspecified
13178 what special meaning is given to any character other than ’&’, ’\’, ’%’, or digits.

13179 A line can be split by substituting a <newline> into it. The application shall ensure it escapes the
13180 <newline> in the *replacement* by preceding it by backslash. Such substitution cannot be done as
13181 part of a **g** or **v** *command list*. The current line number shall be set to the address of the last line
13182 on which a substitution is performed. If no substitution is performed, the current line number
13183 shall be unchanged. If a line is split, a substitution shall be considered to have been performed
13184 on each of the new lines for the purpose of determining the new current line number. A
13185 substitution shall be considered to have been performed even if the replacement string is
13186 identical to the string that it replaces.

13187 The application shall ensure that the value of *flags* is zero or more of:

13188 *count* Substitute for the *count*th occurrence only of the *RE* found on each addressed line.

13189 **g** Globally substitute for all non-overlapping instances of the *RE* rather than just the first
13190 one. If both **g** and *count* are specified, the results are unspecified.

13191 **l** Write to standard output the final line in which a substitution was made. The line shall
13192 be written in the format specified for the **l** command.

13193 **n** Write to standard output the final line in which a substitution was made. The line shall
13194 be written in the format specified for the **n** command.

13195 **p** Write to standard output the final line in which a substitution was made. The line shall
13196 be written in the format specified for the **p** command.

13197 **Copy Command**

13198 *Synopsis*: `(.,.)t`*address*

13199 The **t** command shall be equivalent to the **m** command, except that a copy of the addressed lines
13200 shall be placed after address *address* (which can be 0); the current line number shall be set to the
13201 address of the last line added.

13202 **Undo Command**

13203 *Synopsis*: `u`

13204 The **u** command shall nullify the effect of the most recent command that modified anything in
13205 the buffer, namely the most recent **a**, **c**, **d**, **g**, **i**, **j**, **m**, **r**, **s**, **t**, **u**, **v**, **G**, or **V** command. All changes
13206 made to the buffer by a **g**, **G**, **v**, or **V** global command shall be undone as a single change; if no
13207 changes were made by the global command (such as with **g**/*RE*/**p**), the **u** command shall have
13208 no effect. The current line number shall be set to the value it had immediately before the
13209 command being undone started.

13210 **Global Non-Matched Command**

13211 *Synopsis*: `(1,$)v/RE/command list`

13212 This command shall be equivalent to the global command **g** except that the lines that are marked
13213 during the first step shall be those for which the line excluding the terminating <newline> does
13214 not match the *RE*.

13215 **Interactive Global Not-Matched Command**

13216 *Synopsis*: `(1,$)V/RE/`

13217 This command shall be equivalent to the interactive global command **G** except that the lines that
13218 are marked during the first step shall be those for which the line excluding the terminating
13219 <newline> does not match the *RE*.

13220 **Write Command**

13221 *Synopsis*: `(1,$)w [file]`

13222 The **w** command shall write the addressed lines into the file named by the pathname *file*. The
13223 command shall create the file, if it does not exist, or shall replace the contents of the existing file.
13224 The currently remembered pathname shall not be changed unless there is no remembered
13225 pathname. If no pathname is given, the currently remembered pathname, if any, shall be used
13226 (see the **e** and **f** commands); the current line number shall be unchanged. If the command is
13227 successful, the number of bytes written shall be written to standard output, unless the –**s** option
13228 was specified, in the following format:

13229 `"%d\n", <number of bytes written>`

13230 If *file* begins with `'!'`, the rest of the line shall be taken to be a shell command line whose
13231 standard input shall be the addressed lines. Such a shell command line shall not be remembered
13232 as the current pathname. This usage of the write command with `'!'` shall not be considered as a
13233 ''last **w** command that wrote the entire buffer'', as described previously; thus, this alone shall not
13234 prevent the warning to the user if an attempt is made to destroy the editor buffer via the **e** or **q**
13235 commands.

13236 **Line Number Command**

13237 *Synopsis*: `($)=`

13238 The line number of the addressed line shall be written to standard output in the following
13239 format:

13240 `"%d\n", <line number>`

13241 The current line number shall be unchanged by this command.

13242 **Shell Escape Command**

13243 *Synopsis*: `!command`

13244 The remainder of the line after the `'!'` shall be sent to the command interpreter to be
13245 interpreted as a shell command line. Within the text of that shell command line, the unescaped
13246 character `'%'` shall be replaced with the remembered pathname; if a `'!'` appears as the first
13247 character of the command, it shall be replaced with the text of the previous shell command
13248 executed via `'!'`. Thus, `"!!"` shall repeat the previous !*command*. If any replacements of `'%'` or
13249 `'!'` are performed, the modified line shall be written to the standard output before *command* is
13250 executed. The `'!'` command shall write:

13251 `"!\n"`

13252 to standard output upon completion, unless the –**s** option is specified. The current line number
13253 shall be unchanged.

13254 **Null Command**

13255 *Synopsis*: `(.+1)`

13256 An address alone on a line shall cause the addressed line to be written. A <newline> alone shall
13257 be equivalent to `"+1p"`. The current line number shall be set to the address of the written line.

13258 **EXIT STATUS**

13259 The following exit values shall be returned:

13260 0 Successful completion without any file or command errors.

13261 >0 An error occurred.

13262 **CONSEQUENCES OF ERRORS**

13263 When an error in the input script is encountered, or when an error is detected that is a
13264 consequence of the data (not) present in the file or due to an external condition such as a read or
13265 write error:

13266 • If the standard input is a terminal device file, all input shall be flushed, and a new command
13267 read.

13268 • If the standard input is a regular file, *ed* shall terminate with a non-zero exit status.

13269 **APPLICATION USAGE**

13270 Because of the extremely terse nature of the default error messages, the prudent script writer
13271 begins the *ed* input commands with an **H** command, so that if any errors do occur at least some
13272 clue as to the cause is made available.

13273 In previous versions, an obsolescent – option was described. This is no longer specified.
13274 Applications should use the –**s** option. Using – as a *file* operand now produces unspecified
13275 results. This allows implementations to continue to support the former required behavior.

13276 **EXAMPLES**

13277 None.

13278 **RATIONALE**

13279 The initial description of this utility was adapted from the SVID. It contains some features not
13280 found in Version 7 or BSD-derived systems. Some of the differences between the POSIX and
13281 BSD *ed* utilities include, but need not be limited to:

13282 • The BSD – option does not suppress the `'!'` prompt after a `!` command.

13283 • BSD does not support the special meanings of the `'%'` and `'!'` characters within a `!`
13284 command.

13285 • BSD does not support the *addresses* `';'` and `','`.

13286 • BSD allows the command/suffix pairs **pp**, **ll**, and so on, which are unspecified in this volume
13287 of IEEE Std 1003.1-200x.

13288 • BSD does not support the `'!'` character part of the **e**, **r**, or **w** commands.

13289 • A failed **g** command in BSD sets the line number to the last line searched if there are no
13290 matches.

- BSD does not default the *command list* to the **p** command.
- BSD does not support the **G**, **h**, **H**, **n**, or **V** commands.
- On BSD, if there is no inserted text, the insert command changes the current line to the referenced line –1; that is, the line before the specified line.
- On BSD, the *join* command with only a single address changes the current line to that address.
- BSD does not support the **P** command; moreover, in BSD it is synonymous with the **p** command.
- BSD does not support the *undo* of the commands **j**, **m**, **r**, **s**, or **t**.
- The Version 7 *ed* command **W**, and the BSD *ed* commands **W**, **wq**, and **z** are not present in this volume of IEEE Std 1003.1-200x.

The –**s** option was added to allow the functionality of the now withdrawn – option in a manner compatible with the Utility Syntax Guidelines.

In early proposals there was a limit, {ED_FILE_MAX}, that described the historical limitations of some *ed* utilities in their handling of large files; some of these have had problems with files larger than 100 000 bytes. It was this limitation that prompted much of the desire to include a *split* command in this volume of IEEE Std 1003.1-200x. Since this limit was removed, this volume of IEEE Std 1003.1-200x requires that implementations document the file size limits imposed by *ed* in the conformance document. The limit {ED_LINE_MAX} was also removed; therefore, the global limit {LINE_MAX} is used for input and output lines.

The manner in which the **l** command writes non-printable characters was changed to avoid the historical backspace-overstrike method. On video display terminals, the overstrike is ambiguous because most terminals simply replace overstruck characters, making the **l** format not useful for its intended purpose of unambiguously understanding the content of the line. The historical backslash escapes were also ambiguous. (The string `"a\0011"` could represent a line containing those six characters or a line containing the three characters `'a'`, a byte with a binary value of 1, and a 1.) In the format required here, a backslash appearing in the line is written as `'\\'` so that the output is truly unambiguous. The method of marking the ends of lines was adopted from the *ex* editor and is required for any line ending in <space>s; the `'$'` is placed on all lines so that a real `'$'` at the end of a line cannot be misinterpreted.

Systems with bytes too large to fit into three octal digits must devise other means of displaying non-printable characters. Consideration was given to requiring that the number of octal digits be large enough to hold a byte, but this seemed to be too confusing for applications on the vast majority of systems where three digits are adequate. It would be theoretically possible for the application to use the *getconf* utility to find out the CHAR_BIT value and deal with such an algorithm; however, there is really no portable way that an application can use the octal values of the bytes across various coded character sets, so the additional specification was not worthwhile.

The description of how a NUL is written was removed. The NUL character cannot be in text files, and this volume of IEEE Std 1003.1-200x should not dictate behavior in the case of undefined, erroneous input.

Unlike some of the other editing utilities, the filenames accepted by the **E**, **e**, **R**, and **r** commands are not patterns.

Early proposals stated that the –**p** option worked only when standard input was associated with a terminal device. This has been changed to conform to historical implementations, thereby allowing applications to interpose themselves between a user and the *ed* utility.

13337 The form of the substitute command that uses the **n** suffix was limited in some historical
13338 documentation (where this was described incorrectly as ‘‘backreferencing’’). This limit has been
13339 omitted because there is no reason an editor processing lines of {LINE_MAX} length should have
13340 this restriction. The command **s/x/X/2047** should be able to substitute the 2 047th occurrence of **x**
13341 on a line.

13342 The use of printing commands with printing suffixes (such as **pn**, **lp**, and so on) was made
13343 unspecified because BSD-based systems allow this, whereas System V does not.

13344 Some BSD-based systems exit immediately upon receipt of end-of-file if all of the lines in the file
13345 have been deleted. Since this volume of IEEE Std 1003.1-200x refers to the **q** command in this
13346 instance, such behavior is not allowed.

13347 Some historical implementations returned exit status zero even if command errors had occurred;
13348 this is not allowed by this volume of IEEE Std 1003.1-200x.

13349 Some historical implementations contained a bug that allowed a single period to be entered in
13350 input mode as <backslash> <period> <newline>. This is not allowed by the *ed* because there is
13351 no description of escaping any of the characters in input mode; backslashes are entered into the
13352 buffer exactly as typed. The typical method of entering a single period has been to precede it
13353 with another character and then use the substitute command to delete that character.

13354 It is difficult under some modes of some versions of historical operating system terminal drivers
13355 to distinguish between an end-of-file condition and terminal disconnect. IEEE Std 1003.1-200x
13356 does not require implementations to distinguish between the two situations, which permits
13357 historical implementations of the *ed* utility on historical platforms to conform. Implementations
13358 are encouraged to distinguish between the two, if possible, and take appropriate action on
13359 terminal disconnect.

13360 Historically, *ed* accepted a zero address for the **a** and **r** commands in order to insert text at the
13361 start of the edit buffer. When the buffer was empty the command **.=** returned zero.
13362 IEEE Std 1003.1-200x requires conformance to historical practice.

13363 For consistency with the **a** and **r** commands and better user functionality, the **i** and **c** commands
13364 must also accept an address of 0, in which case 0*i* is treated as 1*i* and likewise for the **c**
13365 command.

13366 All of the following are valid addresses:

13367 `+++`	Three lines after the current line.
13368 `/pattern/-`	One line before the next occurrence of pattern.
13369 `-2`	Two lines before the current line.
13370 `3 ---- 2`	Line one (note the intermediate negative address).
13371 `1 2 3`	Line six.

13372 Any number of addresses can be provided to commands taking addresses; for example,
13373 `"1,2,3,4,5p"` prints lines 4 and 5, because two is the greatest valid number of addresses
13374 accepted by the **print** command. This, in combination with the semicolon delimiter, permits
13375 users to create commands based on ordered patterns in the file. For example, the command
13376 `"3;/foo/;+2p"` will display the first line after line 3 that contains the pattern *foo*, plus the next
13377 two lines. Note that the address `"3;"` must still be evaluated before being discarded, because
13378 the search origin for the `"/foo/"` command depends on this.

13379 Historically, *ed* disallowed address chains, as discussed above, consisting solely of comma or
13380 semicolon separators; for example, `",,,"` or `";;;"` were considered an error. For consistency of
13381 address specification, this restriction is removed. The following table lists some of the address

forms now possible:

Address	Addr1	Addr2	Status	Comment
`7,`	7	7	Historical	
`7,5,`	5	5	Historical	
`7,5,9`	5	9	Historical	
`7,9`	7	9	Historical	
`7,+`	7	8	Historical	
`,`	1	$	Historical	
`,7`	1	7	Extension	
`,,`	$	$	Extension	
`,;`	$	$	Extension	
`7;`	7	7	Historical	
`7;5;`	5	5	Historical	
`7;5;9`	5	9	Historical	
`7;5,9`	5	9	Historical	
`7;$;4`	$	4	Historical	Valid, but erroneous.
`7;9`	7	9	Historical	
`7;+`	7	8	Historical	
`;`	.	$	Historical	
`;7`	.	7	Extension	
`;;`	$	$	Extension	
`;,`	$	$	Extension	

Historically, values could be added to addresses by including them after one or more <blank>s; for example, `"3 – 5p"` wrote the seventh line of the file, and `"/foo/ 5"` was the same as `"5 /foo/"`. However, only absolute values could be added; for example, `"5 /foo/"` was an error. IEEE Std 1003.1-200x requires conformance to historical practice.

Historically, *ed* accepted the `'^'` character as an address, in which case it was identical to the hyphen character. IEEE Std 1003.1-200x does not require or prohibit this behavior.

FUTURE DIRECTIONS

None.

SEE ALSO

ex, *sed*, *sh*, *vi*

CHANGE HISTORY

First released in Issue 2.

Issue 5

In the OPTIONS section, the meaning of –**s** and – is clarified.

Second FUTURE DIRECTION added.

Issue 6

The obsolescent single-minus form has been removed.

A second APPLICATION USAGE note has been added.

The Open Group Corrigendum U025/2 is applied, correcting the description of the Edit section.

The *ed* utility is updated to align with the IEEE P1003.2b draft standard. This includes addition of the treatment of the SIGQUIT signal, changes to *ed* addressing, changes to processing when end-of-file is detected and when terminal disconnect is detected.

13426 The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

13427 **NAME**

13428 env — set the environment for command invocation

13429 **SYNOPSIS**

13430 `env [–i][name=value]... [utility [argument...]]`

13431 **DESCRIPTION**

13432 The *env* utility shall obtain the current environment, modify it according to its arguments, then
13433 invoke the utility named by the *utility* operand with the modified environment.

13434 Optional arguments shall be passed to *utility*.

13435 If no *utility* operand is specified, the resulting environment shall be written to the standard
13436 output, with one *name=value* pair per line.

13437 **OPTIONS**

13438 The *env* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
13439 12.2, Utility Syntax Guidelines.

13440 The following options shall be supported:

13441 **–i** Invoke *utility* with exactly the environment specified by the arguments; the
13442 inherited environment shall be ignored completely.

13443 **OPERANDS**

13444 The following operands shall be supported:

13445 *name=value* Arguments of the form *name=value* shall modify the execution environment, and
13446 shall be placed into the inherited environment before the *utility* is invoked.

13447 *utility* The name of the utility to be invoked. If the *utility* operand names any of the
13448 special built-in utilities in Section 2.14 (on page 2266), the results are undefined.

13449 *argument* A string to pass as an argument for the invoked utility.

13450 **STDIN**

13451 Not used.

13452 **INPUT FILES**

13453 None.

13454 **ENVIRONMENT VARIABLES**

13455 The following environment variables shall affect the execution of *env*:

13456 *LANG* Provide a default value for the internationalization variables that are unset or null.
13457 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
13458 Internationalization Variables for the precedence of internationalization variables
13459 used to determine the values of locale categories.)

13460 *LC_ALL* If set to a non-empty string value, override the values of all the other
13461 internationalization variables.

13462 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
13463 characters (for example, single-byte as opposed to multi-byte characters in
13464 arguments).

13465 *LC_MESSAGES*
13466 Determine the locale that should be used to affect the format and contents of
13467 diagnostic messages written to standard error.

13468 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

PATH Determine the location of the *utility*, as described in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables. If *PATH* is specified as a *name=value* operand to *env*, the *value* given shall be used in the search for *utility*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If no *utility* operand is specified, each *name=value* pair in the resulting environment shall be written in the form:

```
"%s=%s\n", <name>, <value>
```

If the *utility* operand is specified, the *env* utility shall not write to standard output.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

If the *utility* utility is invoked, the exit status of *env* shall be the exit status of *utility*; otherwise, the *env* utility shall exit with one of the following values:

0 The *env* utility completed successfully.

1–125 An error occurred in the *env* utility.

126 The utility specified by *utility* was found but could not be invoked.

127 The utility specified by *utility* could not be found.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if an error occurs so that applications can distinguish ‘‘failure to find a utility’’ from ‘‘invoked utility exited with an error indication’’. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for ‘‘normal error conditions’’ and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for any other reason.

Historical implementations of the *env* utility use the *execvp*( ) or *execlp*( ) functions defined in the System Interfaces volume of IEEE Std 1003.1-200x to invoke the specified utility; this provides better performance and keeps users from having to escape characters with special meaning to the shell. Therefore, shell functions, special built-ins, and built-ins that are only provided by the shell are not found.

EXAMPLES

The following command:

```
env –i PATH=/mybin mygrep xyz myfile
```

invokes the command *mygrep* with a new *PATH* value as the only entry in its environment. In this case, *PATH* is used to locate *mygrep*, which then must reside in **/mybin**.

RATIONALE

As with all other utilities that invoke other utilities, this volume of IEEE Std 1003.1-200x only specifies what *env* does with standard input, standard output, standard error, input files, and output files. If a utility is executed, it is not constrained by the specification of input and output by *env*.

The **–i** option was added to allow the functionality of the withdrawn – option in a manner compatible with the Utility Syntax Guidelines.

Some have suggested that *env* is redundant since the same effect is achieved by:

```
name=value ... utility [ argument ... ]
```

The example is equivalent to *env* when an environment variable is being added to the environment of the command, but not when the environment is being set to the given value. The *env* utility also writes out the current environment if invoked without arguments. There is sufficient functionality beyond what the example provides to justify inclusion of *env*.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5 (on page 2235)

CHANGE HISTORY

First released in Issue 2.

NAME

ex — text editor

SYNOPSIS

UP

```
ex [–rR][–l][–s | –v][–c command][–t tagstring][–w size][file ...]
```

DESCRIPTION

The *ex* utility is a line-oriented text editor. There are two other modes of the editor—open and visual—in which screen-oriented editing is available. This is described more fully by the *ex* **open** and **visual** commands and in *vi*.

This section uses the term *edit buffer* to describe the current working text. No specific implementation is implied by this term. All editing changes are performed on the edit buffer, and no changes to it shall affect any file until an editor command writes the file.

Certain terminals do not have all the capabilities necessary to support the complete *ex* definition, such as the full-screen editing commands (*visual mode* or *open mode*). 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 volume of IEEE Std 1003.1-200x or report an error describing the terminal-related deficiency.

OPTIONS

The *ex* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–c *command* Specify an initial command to be executed in the first edit buffer loaded from an existing file (see the EXTENDED DESCRIPTION section). Implementations may support more than a single **–c** option. In such implementations, the specified commands shall be executed in the order specified on the command line.

–r Recover the named files (see the EXTENDED DESCRIPTION section). Recovery information for a file shall be saved during an editor or system crash (for example, when the editor is terminated by a signal which the editor can catch), or after the use of an *ex* **preserve** 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 number of changes to the edit buffer (since the most recent **preserve** command) that will be recovered is unspecified.

If no *file* operands are given and the **–t** option is not specified, 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 the editor shall exit normally without further action.

–R Set **readonly** edit option.

–s 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 open or visual modes;

see the **visual** command and the description of *vi*.

- Suppress the use of the *EXINIT* environment variable and the reading of any **.exrc** file; see the EXTENDED DESCRIPTION section.
- Suppress autoindentation, ignoring the value of the **autoindent** edit option.

–t *tagstring* Edit the file containing the specified *tagstring*; see *ctags*. The tags feature represented by **–t** *tagstring* and the **tag** command 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. On any system, it shall be an error to specify more than a single **–t** option.

–v Begin in visual mode (see *vi*).

–w *size* Set the value of the *window* editor option to *size*.

OPERANDS

The following operand shall be supported:

file A pathname of a file to be edited.

STDIN

The standard input consists of a series of commands and input text, as described in the EXTENDED DESCRIPTION section. The implementation may limit each line of standard input to a length of {LINE_MAX}.

If the standard input is not a terminal device, it shall be as if the **–s** option had been specified.

If a read from the standard input returns an error, or if the editor detects an end-of-file condition from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

INPUT FILES

Input files shall be text files or files that would be text files except for an incomplete last line that is not longer than {LINE_MAX}–1 bytes in length and contains no NUL characters. By default, any incomplete last line shall be treated as if it had a trailing <newline>. The editing of other forms of files may optionally be allowed by *ex* implementations.

The **.exrc** files and source files shall be text files consisting of *ex* commands; see the EXTENDED DESCRIPTION section.

By default, the editor shall read lines from the files to be edited without interpreting any of those lines as any form of editor command.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *ex*:

COLUMNS Override the system-selected horizontal screen size. See the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables for valid values and results when it is unset or null.

EXINIT Determine a list of *ex* commands that are executed on editor start-up. See the EXTENDED DESCRIPTION section for more details of the initialization phase.

HOME Determine a pathname of a directory that shall be searched for an editor start-up file named **.exrc**; see the EXTENDED DESCRIPTION section.

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

13622 *LC_ALL* If set to a non-empty string value, override the values of all the other
13623 internationalization variables.

13624 *LC_COLLATE*
13625 Determine the locale for the behavior of ranges, equivalence classes, and multi-
13626 character collating elements within regular expressions.

13627 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
13628 characters (for example, single-byte as opposed to multi-byte characters in
13629 arguments and input files), the behavior of character classes within regular
13630 expressions, the classification of characters as uppercase or lowercase letters, the
13631 case conversion of letters, and the detection of word boundaries.

13632 *LC_MESSAGES*
13633 Determine the locale that should be used to affect the format and contents of
13634 diagnostic messages written to standard error.

13635 *LINES* Override the system-selected vertical screen size, used as the number of lines in a
13636 screenful and the vertical screen size in visual mode. See the Base Definitions
13637 volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables for valid values
13638 and results when it is unset or null.

13639 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

13640 *PATH* Determine the search path for the shell command specified in the *ex* editor
13641 commands **!**, **shell**, **read**, and **write**, and the open and visual mode command **!**; see
13642 the description of command search and execution in Section 2.9.1.1 (on page 2249).

13643 *SHELL* Determine the preferred command line interpreter for use as the default value of
13644 the **shell** edit option.

13645 *TERM* Determine the name of the terminal type. If this variable is unset or null, an
13646 unspecified default terminal type shall be used.

13647 **ASYNCHRONOUS EVENTS**

13648 The following term is used in this and following sections to specify command and asynchronous
13649 event actions:

13650 *complete write*
13651 A complete write is a write of the entire contents of the edit buffer to a file of a type
13652 other than a terminal device, or the saving of the edit buffer caused by the user
13653 executing the *ex* **preserve** command. Writing the contents of the edit buffer to a
13654 temporary file that will be removed when the editor exits shall not be considered a
13655 complete write.

13656 The following actions shall be taken upon receipt of signals:

13657 SIGINT If the standard input is not a terminal device, *ex* shall not write the file or return to
13658 command or text input mode, and shall exit with a non-zero exit status.

13659 Otherwise, if executing an open or visual text input mode command, *ex* in receipt
13660 of SIGINT shall behave identically to its receipt of the <ESC> character.

13661 Otherwise:

13662 1. If executing an *ex* text input mode command, all input lines that have been
13663 completely entered shall be resolved into the edit buffer, and any partially
13664 entered line shall be discarded.

2. If there is a currently executing command, it shall be aborted and a message displayed. Unless otherwise specified by the *ex* or *vi* command descriptions, it is unspecified whether any lines modified by the executing command appear modified, or as they were before being modified by the executing command, in the buffer.

   If the currently executing command was a motion command, its associated command shall be discarded.

3. If in open or visual command mode, the terminal shall be alerted.

4. The editor shall then return to command mode.

SIGCONT The screen shall be refreshed if in open or visual mode.

SIGHUP If the edit buffer has been modified since the last complete write, *ex* shall attempt to save the edit buffer so that it can be recovered later using the **–r** option or the *ex* **recover** command. The editor shall not write the file or return to command or text input mode, and shall terminate with a non-zero exit status.

SIGTERM Refer to SIGHUP.

The action taken for all other signals is unspecified.

STDOUT

The standard output shall be used only for writing prompts to the user, for informational messages, and for writing lines from the file.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

The output from *ex* shall be text files.

EXTENDED DESCRIPTION

Only the *ex* mode of the editor is described in this section. See *vi* for additional editing capabilities available in *ex*.

When an error occurs, *ex* shall write a message. If the terminal supports a standout mode (such as inverse video), the message shall be written in standout mode. If the terminal does not support a standout mode, and the edit option **errorbells** is set, an alert action shall precede the error message.

By default, *ex* shall start in command mode, which shall be indicated by a **:** prompt; see the **prompt** command. Text input mode can be entered by the **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.

Initialization in ex and vi

The following symbols are used in this and following sections to specify locations in the edit buffer:

alternate and current path names

Two pathnames, named *current* and *alternate*, are maintained by the editor. Any *ex* commands that take filenames as arguments shall set them as follows:

1. If a *file* argument is specified to the *ex* **edit**, **ex**, or **recover** commands, or if an *ex* **tag** command replaces the contents of the edit buffer.

a. If the command replaces the contents of the edit buffer, the current pathname shall be set to the *file* argument or the file indicated by the tag, and the alternate pathname shall be set to the previous value of the current pathname.

b. Otherwise, the alternate pathname shall be set to the *file* argument.

2. If a *file* argument is specified to the *ex* **next** command:

a. If the command replaces the contents of the edit buffer, the current pathname shall be set to the first *file* argument, and the alternate pathname shall be set to the previous value of the current pathname.

3. If a *file* argument is specified to the *ex* **file** command, the current pathname shall be set to the *file* argument, and the alternate pathname shall be set to the previous value of the current pathname.

4. If a *file* argument is specified to the *ex* **read** and **write** commands (that is, when reading or writing a file, and not to the program named by the **shell** edit option), or a *file* argument is specified to the *ex* **xit** command:

a. If the current pathname has no value, the current pathname shall be set to the *file* argument.

b. Otherwise, the alternate pathname shall be set to the *file* argument.

If the alternate pathname is set to the previous value of the current pathname when the current pathname had no previous value, then the alternate pathname shall have no value as a result.

current line
The line of the edit buffer referenced by the cursor. Each command description specifies the current line after the command has been executed, as the *current line value*. When the edit buffer contains no lines, the current line shall be zero; see **Addressing in ex** (on page 2562).

current column
The current display line column occupied by the cursor. (The columns shall be numbered beginning at 1.) Each command description specifies the current column after the command has been executed, as the *current column* value. This column is an *ideal* column that is remembered over the lifetime of the editor. The actual display line column upon which the cursor rests may be different from the current column; see the cursor positioning discussion in **Command Descriptions in vi** (on page 3186).

set to non-<blank>
A description for a current column value, meaning that the current column shall be set to the last display line column on which is displayed any part of the first non-<blank> of the line. If the line has no non-<blank> non- <newline>s, the current column shall be set to the last display line column on which is displayed any part of the last non-<newline> in the line. If the line is empty, the current column shall be set to column position 1.

The length of lines in the edit buffer may be limited to {LINE_MAX} bytes. In open and visual mode, the length of lines in the edit buffer may be limited to the number of characters that will fit in the display. If either limit is exceeded during editing, an error message shall be written. If either limit is exceeded by a line read in from a file, an error message shall be written and the edit session may be terminated.

If the editor stops running due to any reason other than a user command, and the edit buffer has been modified since the last complete write, it shall be equivalent to a SIGHUP asynchronous event. If the system crashes, it shall be equivalent to a SIGHUP asynchronous event.

13752 During initialization (before the first file is copied into the edit buffer or any user commands
13753 from the terminal are processed) the following shall occur:

13754 1. If the environment variable *EXINIT* is set, the editor shall execute the *ex* commands
13755 contained in that variable.

13756 2. If the *EXINIT* variable is not set, and all of the following are true:

13757 a. The *HOME* environment variable is not null and not empty.

13758 b. The file **.exrc** in the directory referred to by the *HOME* environment variable:

13759 1. Exists

13760 2. Is owned by the same user ID as the real user ID of the process or the process
13761 has appropriate privileges

13762 3. Is not writeable by anyone other than the owner

13763 the editor shall execute the *ex* commands contained in that file.

13764 3. If and only if all the following are true:

13765 a. The current directory is not referred to by the *HOME* environment variable.

13766 b. A command in the *EXINIT* environment variable or a command in the **.exrc** file in the
13767 directory referred to by the *HOME* environment variable sets the editor option **exrc**.

13768 c. The **.exrc** file in the current directory:

13769 1. Exists

13770 2. Is owned by the same user ID as the real user ID of the process, or by one of a
13771 set of implementation-defined user IDs

13772 3. Is not writeable by anyone other than the owner

13773 the editor shall attempt to execute the *ex* commands contained in that file.

13774 Lines in any **.exrc** file that are blank lines shall be ignored. If any **.exrc** file exists, but is not read
13775 for ownership or permission reasons, it shall be an error.

13776 After the *EXINIT* variable and any **.exrc** files are processed, the first file specified by the user
13777 shall be edited, as follows:

13778 1. If the user specified the **–t** option, the effect shall be as if the *ex* **tag** command was entered
13779 with the specified argument, with the exception that if tag processing does not result in a
13780 file to edit, the effect shall be as described in step 3. below.

13781 2. Otherwise, if the user specified any command line *file* arguments, the effect shall be as if
13782 the *ex* **edit** command was entered with the first of those arguments as its *file* argument.

13783 3. Otherwise, the effect shall be as if the *ex* **edit** command was entered with a nonexistent
13784 filename as its *file* argument. It is unspecified whether this action shall set the current
13785 pathname. In an implementation where this action does not set the current pathname, any
13786 editor command using the current pathname shall fail until an editor command sets the
13787 current pathname.

13788 If the **–r** option was specified, the first time a file in the initial argument list or a file specified by
13789 the **–t** option is edited, if recovery information has previously been saved about it, that
13790 information shall be recovered and the editor shall behave as if the contents of the edit buffer
13791 have already been modified. If there are multiple instances of the file to be recovered, the one
13792 most recently saved shall be recovered, and an informational message that there are previous

versions of the file that can be recovered shall be written. If no recovery information about a file is available, an informational message to this effect shall be written, and the edit shall proceed as usual.

If the **–c** option was specified, the first time a file that already exists (including a file that might not exist but for which recovery information is available, when the **–r** option is specified) replaces or initializes the contents of the edit buffer, the current line shall be set to the last line of the edit buffer, the current column shall be set to non-<blank>, and the *ex* commands specified with the **–c** option shall be executed. In this case, the current line and current column shall not be set as described for the command associated with the replacement or initialization of the edit buffer contents. However, if the **–t** option or a **tag** command is associated with this action, the **–c** option commands shall be executed and then the movement to the tag shall be performed.

The current argument list shall initially be set to the filenames specified by the user on the command line. If no filenames are specified by the user, the current argument list shall be empty. If the **–t** option was specified, it is unspecified whether any filename resulting from tag processing shall be prepended to the current argument list. In the case where the filename is added as a prefix to the current argument list, the current argument list reference shall be set to that filename. In the case where the filename is not added as a prefix to the current argument list, the current argument list reference shall logically be located before the first of the filenames specified on the command line (for example, a subsequent *ex* **next** command shall edit the first filename from the command line). If the **–t** option was not specified, the current argument list reference shall be to the first of the filenames on the command line.

Addressing in ex

Addressing in *ex* relates to the current line and the current column; the address of a line is its 1-based line number, the address of a column is its 1-based count from the beginning of the line. Generally, the current line is the last line affected by a command. The current line number is the address of the current line. In each command description, the effect of the command on the current line number and the current column is described.

Addresses are constructed as follows:

1. The character `'.'` (period) shall address the current line.
2. The character `'$'` shall address the last line of the edit buffer.
3. The positive decimal number *n* shall address the *n*th line of the edit buffer.
4. The address `"'x"` refers to the line marked with the mark name character `'x'`, which shall be a lowercase letter from the portable character set or one of the characters `''` or `'`. It shall be an error if the line that was marked is not currently present in the edit buffer or the mark has not been set. Lines can be marked with the *ex* **mark** or **k** commands, or the *vi* **m** command.
5. A regular expression (RE) enclosed by slashes (`'/'`) shall address the first line found by searching forwards from the line following the current line toward the end of the edit buffer and stopping at the first line for which the line excluding the terminating <newline> matches the regular expression. As stated in **Regular Expressions in ex** (on page 2592), an address consisting of a null regular expression delimited by slashes `"//"` shall address the next line for which the line excluding the terminating <newline> matches the last regular expression encountered. In addition, the second slash can be omitted at the end of a command line. If the **wrapscan** edit option is set, the search shall wrap around to the beginning of the edit buffer and continue up to and including the current line, so that the entire edit buffer is searched. Within the regular expression, the sequence `"\/"` shall represent a literal slash instead of the regular expression delimiter.

6. A regular expression enclosed in question marks (`'?'`) shall address the first line found by searching backwards from the line preceding the current line toward the beginning of the edit buffer and stopping at the first line for which the line excluding the terminating <newline> matches the regular expression. An address consisting of a null regular expression delimited by question marks `"??"` shall address the previous line for which the line excluding the terminating <newline> matches the last regular expression encountered. In addition, the second question mark can be omitted at the end of a command line. If the **wrapscan** edit option is set, the search shall wrap around from the beginning of the edit buffer to the end of the edit buffer and continue up to and including the current line, so that the entire edit buffer is searched. Within the regular expression, the sequence `"\?"` shall represent a literal question mark instead of the RE delimiter.

7. A plus sign (`'+'`) or a minus sign (`'-'`) followed by a decimal number shall address the current line plus or minus the number. A `'+'` or `'-'` not followed by a decimal number shall address the current line plus or minus 1.

Addresses can be followed by zero or more address offsets, optionally <blank>-separated. Address offsets are constructed as follows:

1. A `'+'` or `'-'` immediately followed by a decimal number shall add (subtract) the indicated number of lines to (from) the address. A `'+'` or `'-'` not followed by a decimal number shall add (subtract) 1 to (from) the address.

2. A decimal number shall add the indicated number of lines to the address.

It shall not be an error for an intermediate address value to be less than zero or greater than the last line in the edit buffer. It shall be an error for the final address value to be less than zero or greater than the last line in the edit buffer.

Commands take zero, one, or two addresses; see the descriptions of *1addr* and *2addr* in **Command Descriptions in ex** (on page 2569). If more than the required number of addresses are provided to a command that requires zero addresses, it shall be an error. Otherwise, if more than the required number of addresses are provided to a command, the addresses specified first shall be evaluated and then discarded until the maximum number of valid addresses remain.

Addresses shall be separated from each other by a comma (`','`) or a semicolon (`';'`). If no address is specified before or after a comma or semicolon separator, it shall be as if the address of the current line was specified before or after the separator. In the case of a semicolon separator, the current line (`'.'`) shall be set to the first address, and only then will the next address be calculated. This feature can be used to determine the starting line for forwards and backwards searches (see rules 5. and 6.).

A percent sign (`'%'`) shall be equivalent to entering the two addresses `"1,$"`.

Any delimiting <blank>s between addresses, address separators, or address offsets shall be discarded.

Command Line Parsing in ex

The following symbol is used in this and following sections to describe parsing behavior:

escape If a character is referred to as ‘‘backslash escaped’’ or ‘‘<control>-V escaped,’’ it shall mean that the character acquired or lost a special meaning by virtue of being preceded, respectively, by a backslash or <control>-V character. Unless otherwise specified, the escaping character shall be discarded at that time and shall not be further considered for any purpose.

Command-line parsing shall be done in the following steps. For each step, characters already evaluated shall be ignored; that is, the phrase ‘‘leading character’’ refers to the next character that has not yet been evaluated.

1. Leading colon characters shall be skipped.
2. Leading <blank>s shall be skipped.
3. If the leading character is a double-quote character, the characters up to and including the next non-backslash-escaped <newline> shall be discarded, and any subsequent characters shall be parsed as a separate command.
4. Leading characters that can be interpreted as addresses shall be evaluated; see **Addressing in ex** (on page 2562).
5. Leading <blank>s shall be skipped.
6. If the next character is a vertical-line character or a <newline>:
   a. If the next character is a <newline>:
      1. If *ex* is in open or visual mode, the current line shall be set to the last address specified, if any.
      2. Otherwise, if the last command was terminated by a vertical-line character, no action shall be taken; for example, the command `"||<newline>"` shall execute two implied commands, not three.
      3. Otherwise, step 6.b. shall apply.
   b. Otherwise, the implied command shall be the **print** command. The last **#**, **p**, and **l** flags specified to any *ex* command shall be remembered and shall apply to this implied command. Executing the *ex* **number**, **print**, or **list** command shall set the remembered flags to **#**, nothing, and **l**, respectively, plus any other flags specified for that execution of the **number**, **print**, or **list** command.

      If *ex* is not currently performing a **global** or **v** command, and no address or count is specified, the current line shall be incremented by 1 before the command is executed. If incrementing the current line would result in an address past the last line in the edit buffer, the command shall fail, and the increment shall not happen.
   c. The <newline> or vertical-line character shall be discarded and any subsequent characters shall be parsed as a separate command.
7. The command name shall be comprised of the next character (if the character is not alphabetic), or the next character and any subsequent alphabetic characters (if the character is alphabetic), with the following exceptions:
   a. Commands that consist of any prefix of the characters in the command name **delete**, followed immediately by any of the characters `'l'`, `'p'`, `'+'`, `'-'`, or `'#'` shall be interpreted as a **delete** command, followed by a <blank>, followed by the characters that were not part of the prefix of the **delete** command. The maximum number of characters shall be matched to the command name **delete**; for example, `"del"` shall not be treated as `"de"` followed by the flag **l**.
   b. Commands that consist of the character `'k'`, followed by a character that can be used as the name of a mark, shall be equivalent to the mark command followed by a <blank>, followed by the character that followed the `'k'`.
   c. Commands that consist of the character `'s'`, followed by characters that could be interpreted as valid options to the **s** command, shall be the equivalent of the **s**

13928 command, without any pattern or replacement values, followed by a <blank>,
13929 followed by the characters after the 's'.

13930 8. The command name shall be matched against the possible command names, and a
13931 command name that contains a prefix matching the characters specified by the user shall
13932 be the executed command. In the case of commands where the characters specified by the
13933 user could be ambiguous, the executed command shall be as follows:

a	**append**	**n**	**next**	**t**	**t**
c	**change**	**p**	**print**	**u**	**undo**
ch	**change**	**pr**	**print**	**un**	**undo**
e	**edit**	**r**	**read**	**v**	**v**
m	**move**	**re**	**read**	**w**	**write**
ma	**mark**	**s**	**s**		

13940 Implementation extensions with names causing similar ambiguities shall not be checked
13941 for a match until all possible matches for commands specified by IEEE Std 1003.1-200x
13942 have been checked.

13943 9. If the command is a **!** command, or if the command is a **read** command followed by zero
13944 or more <blank>s and a **!**, or if the command is a **write** command followed by one or more
13945 <blank>s and a **!**, the rest of the command shall include all characters up to a non-
13946 backslash-escaped <newline>. The <newline> shall be discarded and any subsequent
13947 characters shall be parsed as a separate *ex* command.

13948 10. Otherwise, if the command is an **edit**, **ex**, or **next** command, or a **visual** command while in
13949 open or visual mode, the next part of the command shall be parsed as follows:

13950 a. Any '!' character immediately following the command shall be skipped and be part
13951 of the command.

13952 b. Any leading <blank>s shall be skipped and be part of the command.

13953 c. If the next character is a '+', characters up to the first non-backslash-escaped
13954 <newline> or non-backslash-escaped <blank> shall be skipped and be part of the
13955 command.

13956 d. The rest of the command shall be determined by the steps specified in paragraph 12.

13957 11. Otherwise, if the command is a **global**, **open**, **s**, or **v** command, the next part of the
13958 command shall be parsed as follows:

13959 a. Any leading <blank>s shall be skipped and be part of the command.

13960 b. If the next character is not an alphanumeric, double-quote, <newline>, backslash, or
13961 vertical-line character:

13962 1. The next character shall be used as a command delimiter.

13963 2. If the command is a **global**, **open**, or **v** command, characters up to the first
13964 non-backslash-escaped <newline>, or first non-backslash-escaped delimiter
13965 character, shall be skipped and be part of the command.

13966 3. If the command is an **s** command, characters up to the first non-backslash-
13967 escaped <newline>, or second non-backslash-escaped delimiter character, shall
13968 be skipped and be part of the command.

13969 c. If the command is a **global** or **v** command, characters up to the first non-backslash-
13970 escaped <newline> shall be skipped and be part of the command.

13971 d. Otherwise, the rest of the command shall be determined by the steps specified in
13972 paragraph 12.

13973 12. Otherwise:

13974 a. If the command was a **map**, **unmap**, **abbreviate**, or **unabbreviate** command,
13975 characters up to the first non-<control>-V-escaped <newline>, vertical-line, or
13976 double-quote character shall be skipped and be part of the command.

13977 b. Otherwise, characters up to the first non-backslash-escaped <newline>, vertical-line,
13978 or double-quote character shall be skipped and be part of the command.

13979 c. If the command was an **append**, **change**, or **insert** command, and the step 12.b.
13980 ended at a vertical-line character, any subsequent characters, up to the next non-
13981 backslash-escaped <newline> shall be used as input text to the command.

13982 d. If the command was ended by a double-quote character, all subsequent characters,
13983 up to the next non-backslash-escaped <newline>, shall be discarded.

13984 e. The terminating <newline> or vertical-line character shall be discarded and any
13985 subsequent characters shall be parsed as a separate *ex* command.

13986 Command arguments shall be parsed as described by the Synopsis and Description of each
13987 individual *ex* command. This parsing shall not be <blank>-sensitive, except for the **!** argument,
13988 which must follow the command name without intervening <blank>s, and where it would
13989 otherwise be ambiguous. For example, *count* and *flag* arguments need not be <blank>-separated
13990 because `"d22p"` is not ambiguous, but *file* arguments to the *ex* **next** command must be
13991 separated by one or more <blank>s. Any <blank> in command arguments for the **abbreviate**,
13992 **unabbreviate**, **map**, and **unmap** commands can be <control>-V-escaped, in which case the
13993 <blank> shall not be used as an argument delimiter. Any <blank> in the command argument for
13994 any other command can be backslash-escaped, in which case that <blank> shall not be used as
13995 an argument delimiter.

13996 Within command arguments for the **abbreviate**, **unabbreviate**, **map**, and **unmap** commands,
13997 any character can be <control>-V-escaped. All such escaped characters shall be treated literally
13998 and shall have no special meaning. Within command arguments for all other *ex* commands that
13999 are not regular expressions or replacement strings, any character that would otherwise have a
14000 special meaning can be backslash-escaped. Escaped characters shall be treated literally, without
14001 special meaning as shell expansion characters or `'!'`, `'%'`, and `'#'` expansion characters. See
14002 **Regular Expressions in ex** (on page 2592) and **Replacement Strings in ex** (on page 2592) for
14003 descriptions of command arguments that are regular expressions or replacement strings.

14004 Non-backslash-escaped `'%'` characters appearing in *file* arguments to any *ex* command shall be
14005 replaced by the current pathname; unescaped `'#'` characters shall be replaced by the alternate
14006 pathname. It shall be an error if `'%'` or `'#'` characters appear unescaped in an argument and
14007 their corresponding values are not set.

14008 Non-backslash-escaped `'!'` characters in the arguments to either the *ex* **!** command or the open
14009 and visual mode **!** command, or in the arguments to the *ex* **read** command, where the first non-
14010 <blank> after the command name is a `'!'` character, or in the arguments to the *ex* **write**
14011 command where the command name is followed by one or more <blank>s and the first non-
14012 <blank> after the command name is a `'!'` character, shall be replaced with the arguments to the
14013 last of those three commands as they appeared after all unescaped `'%'`, `'#'`, and `'!'` characters
14014 were replaced. It shall be an error if `'!'` characters appear unescaped in one of these commands
14015 and there has been no previous execution of one of these commands.

14016 If an error occurs during the parsing or execution of an *ex* command:

- An informational message to this effect shall be written. Execution of the *ex* command shall stop, and the cursor (for example, the current line and column) shall not be further modified.
- If the *ex* command resulted from a map expansion, all characters from that map expansion shall be discarded, except as otherwise specified by the **map** command.
- Otherwise, if the *ex* command resulted from the processing of an *EXINIT* environment variable, a **.exrc** file, a **:source** command, a **–c** option, or a **+***command* specified to an *ex* **edit**, **ex**, **next**, or **visual** command, no further commands from the source of the commands shall be executed.
- Otherwise, if the *ex* command resulted from the execution of a buffer or a **global** or **v** command, no further commands caused by the execution of the buffer or the **global** or **v** command shall be executed.
- Otherwise, if the *ex* command was not terminated by a <newline>, all characters up to and including the next non-backslash-escaped <newline> shall be discarded.

Input Editing in ex

The following symbols are used in this and following sections to specify command actions.

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 edit buffer.

When accepting input characters from the user, in either *ex* command mode or *ex* text input mode, *ex* shall enable canonical mode input processing, as defined in the System Interfaces volume of IEEE Std 1003.1-200x.

If in *ex* text input mode:

1. If the **number** edit option is set, *ex* shall prompt for input using the line number that would be assigned to the line if it is entered, in the format specified for the *ex* **number** command.
2. If the **autoindent** edit option is set, *ex* shall prompt for input using **autoindent** characters, as described by the **autoindent** edit option. **autoindent** characters shall follow the line number, if any.

If in *ex* command mode:

1. If the **prompt** edit option is set, input shall be prompted for using a single `':'` character; otherwise, there shall be no prompt.

The input characters in the following sections shall have the following effects on the input line.

Scroll

Synopsis: `eof`

See the description of the *stty eof* character in *stty*.

If in *ex* command mode:

If the *eof* character is the first character entered on the line, the line shall be evaluated as if it contained two characters: a <control>-D and a <newline>.

Otherwise, the *eof* character shall have no special meaning.

If in *ex* text input mode:

If the cursor follows an **autoindent** character, the **autoindent** characters in the line shall be modified so that a part of the next text input character will be displayed on the first column in the line after the previous **shiftwidth** edit option column boundary, and the user shall be prompted again for input for the same line.

Otherwise, if the cursor follows a `'0'`, which follows an **autoindent** character, and the `'0'` was the previous text input character, the `'0'` and all **autoindent** characters in the line shall be discarded, and the user shall be prompted again for input for the same line.

Otherwise, if the cursor follows a `'^'`, which follows an **autoindent** character, and the `'^'` was the previous text input character, the `'^'` and all **autoindent** characters in the line shall be discarded, and the user shall be prompted again for input for the same line. In addition, the **autoindent** level for the next input line shall be derived from the same line from which the **autoindent** level for the current input line was derived.

Otherwise, if there are no **autoindent** or text input characters in the line, the *eof* character shall be discarded.

Otherwise, the *eof* character shall have no special meaning.

<newline>

Synopsis: `<newline>`
`<control>-J`

If in *ex* command mode:

Cause the command line to be parsed; <control>-J shall be mapped to the <newline> for this purpose.

If in *ex* text input mode:

Terminate the current line. If there are no characters other than **autoindent** characters on the line, all characters on the line shall be discarded.

Prompt for text input on a new line after the current line. If the **autoindent** edit option is set, an appropriate number of **autoindent** characters shall be added as a prefix to the line as described by the *ex* **autoindent** edit option.

<backslash>

Synopsis: `<backslash>`

Allow the entry of a subsequent <newline> or <control>-J as a literal character, removing any special meaning that it may have to the editor during text input mode. The backslash character shall be retained and evaluated when the command line is parsed, or retained and included when the input text becomes part of the edit buffer.

<control>-V

Synopsis: `<control>-V`

Allow the entry of any subsequent character as a literal character, removing any special meaning that it may have to the editor during text input mode. The <control>-V character shall be discarded before the command line is parsed or the input text becomes part of the edit buffer.

If the ‘‘literal next’’ functionality is performed by the underlying system, it is implementation-defined whether a character other than <control>-V performs this function.

<control>-W

Synopsis: `<control>-W`

Discard the <control>-W, and the word previous to it in the input line, including any <blank>s following the word and preceding the <control>-W. If the ‘‘word erase’’ functionality is performed by the underlying system, it is implementation-defined whether a character other than <control>-W performs this function.

Command Descriptions in ex

The following symbols are used in this section to represent command modifiers. Some of these modifiers can be omitted, in which case the specified defaults shall be used.

1addr A single line address, given in any of the forms described in **Addressing in ex** (on page 2562); the default shall be the current line (`'.'`), unless otherwise specified.

If the line address is zero, it shall be an error, unless otherwise specified in the following command descriptions.

If the edit buffer is empty, and the address is specified with a command other than **=**, **append**, **insert**, **open**, **put**, **read**, or **visual**, or the address is not zero, it shall be an error.

2addr Two addresses specifying an inclusive range of lines. If no addresses are specified, the default for *2addr* shall be the current line only (`"."`), unless otherwise specified in the following command descriptions. If one address is specified, *2addr* shall specify that line only, unless otherwise specified in the following command descriptions.

It shall be an error if the first address is greater than the second address.

If the edit buffer is empty, and the two addresses are specified with a command other than the **!**, **write**, **wq**, or **xit** commands, or either address is not zero, it shall be an error.

count A positive decimal number. If *count* is specified, it shall be equivalent to specifying an additional address to the command, unless otherwise specified by the following command descriptions. The additional address shall be equal to the last address specified to the command (either explicitly or by default) plus *count*−1.

If this would result in an address greater than the last line of the edit buffer, it shall be corrected to equal the last line of the edit buffer.

flags One or more of the characters `'+'`, `'−'`, `'#'`, `'p'`, or `'l'` (ell). The flag characters can be <blank>-separated, and in any order or combination. The characters `'#'`, `'p'`, and `'l'` shall cause lines to be written in the format specified by the **print** command with the specified *flags*.

The lines to be written are as follows:

1. All edit buffer lines written during the execution of the *ex* **&**, **~**, **list**, **number**, **open**, **print**, **s**, **visual**, and **z** commands shall be written as specified by *flags*.
2. After the completion of an *ex* command with a flag as an argument, the current line shall be written as specified by *flags*, unless the current line was the last line written by the command.

The characters `'+'` and `'−'` cause the value of the current line after the execution of the *ex* command to be adjusted by the offset address as described in **Addressing**

in ex (on page 2562). This adjustment shall occur before the current line is written as described in 2. above.

The default for *flags* shall be none.

buffer One of a number of named areas for holding text. The named buffers are specified by the alphanumeric characters of the POSIX locale. There shall also be one ‘‘unnamed’’ buffer. When no buffer is specified for editor commands that use a buffer, the unnamed buffer shall be used. Commands that store text into buffers shall store the text as it was before the command took effect, and shall store text occurring earlier in the file before text occurring later in the file, regardless of how the text region was specified. Commands that store text into buffers shall store the text into the unnamed buffer as well as any specified buffer.

In *ex* commands, buffer names are specified as the name by itself. In open or visual mode commands the name is preceded by a double quote (`'"'`) character.

If the specified buffer name is an uppercase character, and the buffer contents are to be modified, the buffer shall be appended to rather than being overwritten. If the buffer is not being modified, specifying the buffer name in lowercase and uppercase shall have identical results.

There shall also be buffers named by the numbers 1 through 9. In open and visual mode, if a region of text including characters from more than a single line is being modified by the *vi* **c** or **d** commands, the motion character associated with the **c** or **d** commands specifies that the buffer text shall be in line mode, or the commands **%**, **‘**, **/**, **?**, **(**, **)**, **N**, **n**, **{**, or **}** are used to define a region of text for the **c** or **d** commands, the contents of buffers 1 through 8 shall be moved into the buffer named by the next numerically greater value, the contents of buffer 9 shall be discarded, and the region of text shall be copied into buffer 1. This shall be in addition to copying the text into a user-specified buffer or unnamed buffer, or both. Numeric buffers can be specified as a source buffer for open and visual mode commands; however, specifying a numeric buffer as the write target of an open or visual mode command shall have unspecified results.

The text of each buffer shall have the characteristic of being in either line or character mode. Appending text to a non-empty buffer shall set the mode to match the characteristic of the text being appended. Appending text to a buffer shall cause the creation of at least one additional line in the buffer. All text stored into buffers by *ex* commands shall be in line mode. The *ex* commands that use buffers as the source of text specify individually how buffers of different modes are handled. Each open or visual mode command that uses buffers for any purpose specifies individually the mode of the text stored into the buffer and how buffers of different modes are handled.

file Command text used to derive a pathname. The default shall be the current pathname, as defined previously, in which case, if no current pathname has yet been established it shall be an error, except where specifically noted in the individual command descriptions that follow. If the command text contains any of the characters `'~'`, `'{'`, `'['`, `'*'`, `'?'`, `'$'`, `'''`, `'''`, `'"'`, and `'\'`, it shall be subjected to the process of ‘‘shell expansions’’, as described below; if more than a single pathname results and the command expects only one, it shall be an error.

The process of shell expansions in the editor shall be done as follows. The *ex* utility shall pass two arguments to the program named by the shell edit option; the first shall be **–c**, and the second shall be the string `"echo"` and the command text as a

single argument. The standard output and standard error of that command shall replace the command text.

! A character that can be appended to the command name to modify its operation, as detailed in the individual command descriptions. With the exception of the *ex* **read**, **write**, and **!** commands, the '!' character shall only act as a modifier if there are no <blank>s between it and the command name.

remembered search direction
The *vi* commands **N** and **n** begin searching in a forwards or backwards direction in the edit buffer based on a remembered search direction, which is initially unset, and is set by the *ex* **global**, **v**, **s**, and **tag** commands, and the *vi* **/** and **?** commands.

Abbreviate

Synopsis: `ab[`*`breviate`*`][`*`lhs rhs`*`]`

If *lhs* and *rhs* are not specified, write the current list of abbreviations and do nothing more.

Implementations may restrict the set of characters accepted in *lhs* or *rh*, except that printable characters and <blank>s shall not be restricted. Additional restrictions shall be implementation-defined.

In both *lhs* and *rhs*, any character may be escaped with a <control>-V, in which case the character shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V shall be discarded.

In open and visual text input mode, if a non-word or <ESC> character that is not escaped by a <control>-V character is entered after a word character, a check shall be made for a set of characters matching *lhs*, in the text input entered during this command. If it is found, the effect shall be as if *rhs* was entered instead of *lhs*.

The set of characters that are checked is defined as follows:

1. If there are no characters inserted before the word and non-word or <ESC> characters that triggered the check, the set of characters shall consist of the word character.
2. If the character inserted before the word and non-word or <ESC> characters that triggered the check is a word character, the set of characters shall consist of the characters inserted immediately before the triggering characters that are word characters, plus the triggering word character.
3. If the character inserted before the word and non-word or <ESC> characters that triggered the check is not a word character, the set of characters shall consist of the characters that were inserted before the triggering characters that are neither <blank>s nor word characters, plus the triggering word character.

It is unspecified whether the *lhs* argument entered for the *ex* **abbreviate** and **unabbreviate** commands is replaced in this fashion. Regardless of whether or not the replacement occurs, the effect of the command shall be as if the replacement had not occurred.

Current line: Unchanged.

Current column: Unchanged.

14226 **Append**

14227 *Synopsis*: **[***1addr***]** a**[**ppend**][**!**]**

14228 Enter text input mode; the input text shall be placed after the specified line. If line zero is
14229 specified, the text shall be placed at the beginning of the edit buffer.

14230 This command shall be affected by the **number** and **autoindent** edit options; following the
14231 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the
14232 duration of this command only.

14233 *Current line*: Set to the last input line; if no lines were input, set to the specified line, or to the
14234 first line of the edit buffer if a line of zero was specified, or zero if the edit buffer is empty.

14235 *Current column*: Set to non-<blank>.

14236 **Arguments**

14237 *Synopsis*: ar**[***gs***]**

14238 Write the current argument list, with the current argument-list entry, if any, between '[' and
14239 ']' characters.

14240 *Current line*: Unchanged.

14241 *Current column*: Unchanged.

14242 **Change**

14243 *Synopsis*: **[***2addr***]** c**[**hange**][**!**][***count***]**

14244 Enter *ex* text input mode; the input text shall replace the specified lines. The specified lines shall
14245 be copied into the unnamed buffer, which shall become a line mode buffer.

14246 This command shall be affected by the **number** and **autoindent** edit options; following the
14247 command name with '!' shall cause the **autoindent** edit option setting to be toggled for the
14248 duration of this command only.

14249 *Current line*: Set to the last input line; if no lines were input, set to the line before the first
14250 address, or to the first line of the edit buffer if there are no lines preceding the first address, or to
14251 zero if the edit buffer is empty.

14252 *Current column*: Set to non-<blank>.

14253 **Change Directory**

14254 *Synopsis*: chd**[**ir**][**!**][***directory***]**
14255 cd**[**!**][***directory***]**

14256 Change the current working directory to *directory*.

14257 If no *directory* argument is specified, and the *HOME* environment variable is set to a non-null
14258 and non-empty value, *directory* shall default to the value named in the *HOME* environment
14259 variable. If the *HOME* environment variable is empty or is undefined, the default value of
14260 *directory* is implementation-defined.

14261 If no '!' is appended to the command name, and the edit buffer has been modified since the
14262 last complete write, and the current pathname does not begin with a '/', it shall be an error.

14263 *Current line*: Unchanged.

Current column: Unchanged.

Copy

Synopsis: `[`*2addr*`] co[py] `*1addr*` [`*flags*`]`
`[`*2addr*`] t `*1addr*` [`*flags*`]`

Copy the specified lines after the specified destination line; line zero specifies that the lines shall be placed at the beginning of the edit buffer.

Current line: Set to the last line copied.

Current column: Set to non-<blank>.

Delete

Synopsis: `[`*2addr*`] d[elete][`*buffer*`][`*count*`][`*flags*`]`

Delete the specified lines into a buffer (defaulting to the unnamed buffer), which shall become a line-mode buffer.

Flags can immediately follow the command name; see **Command Line Parsing in ex** (on page 2563).

Current line: Set to the line following the deleted lines, or to the last line in the edit buffer if that line is past the end of the edit buffer, or to zero if the edit buffer is empty.

Current column: Set to non-<blank>.

Edit

Synopsis: `e[dit][!][+`*command*`][`*file*`]`
`ex[!][+`*command*`][`*file*`]`

If no '!' is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error.

If *file* is specified, replace the current contents of the edit buffer with the current contents of *file*, and set the current pathname to *file*. If *file* is not specified, replace the current contents of the edit buffer with the current contents of the file named by the current pathname. If for any reason the current contents of the file cannot be accessed, the edit buffer shall be empty.

The **+***command* option shall be <blank>-delimited; <blank>s within **+***command* can be escaped by preceding them with a backslash character. The **+***command* shall be interpreted as an *ex* command immediately after the contents of the edit buffer have been replaced and the current line and column have been set.

If the edit buffer is empty:

Current line: Set to 0.

Current column: Set to 1.

Otherwise, if executed while in *ex* command mode or if the **+***command* argument is specified:

Current line: Set to the last line of the edit buffer.

Current column: Set to non-<blank>.

Otherwise, if *file* is omitted or results in the current pathname:

Current line: Set to the first line of the edit buffer.

Current column: Set to non-<blank>.

Otherwise, if *file* is the same as the last file edited, the line and column shall be set as follows; if the file was previously edited, the line and column may be set as follows:

Current line: Set to the last value held when that file was last edited. If this value is not a valid line in the new edit buffer, set to the first line of the edit buffer.

Current column: If the current line was set to the last value held when the file was last edited, set to the last value held when the file was last edited. Otherwise, or if the last value is not a valid column in the new edit buffer, set to non-<blank>.

Otherwise:

Current line: Set to the first line of the edit buffer.

Current column: Set to non-<blank>.

File

Synopsis: `f[ile][`*`file`*`]`

If a *file* argument is specified, the alternate pathname shall be set to the current pathname, and the current pathname shall be set to *file*.

Write an informational message. If the file has a current pathname, it shall be included in this message; otherwise, the message shall indicate that there is no current pathname. If the edit buffer contains lines, the current line number and the number of lines in the edit buffer shall be included in this message; otherwise, the message shall indicate that the edit buffer is empty. If the edit buffer has been modified since the last complete write, this fact shall be included in this message. If the **readonly** edit option is set, this fact shall be included in this message. The message may contain other unspecified information.

Current line: Unchanged.

Current column: Unchanged.

Global

Synopsis: `[`*`2addr`*`] g[lobal] /`*`pattern`*`/ [`*`commands`*`]`
`[`*`2addr`*`] v /`*`pattern`*`/ [`*`commands`*`]`

The optional `'!'` character after the **global** command shall be the same as executing the **v** command.

If *pattern* is empty (for example, "`//`") or not specified, the last regular expression used in the editor command shall be used as the *pattern*. The *pattern* can be delimited by slashes (shown in the Synopsis), as well as any non-alphanumeric or non-<blank> other than backslash, vertical line, double quote, or <newline>.

If no lines are specified, the lines shall default to the entire file.

The **global** and **v** commands are logically two-pass operations. First, mark the lines within the specified lines for which the line excluding the terminating <newline> matches (**global**) or does not match (**v** or **global!**) the specified pattern. Second, execute the *ex* commands given by *commands*, with the current line (`'.'`) set to each marked line. If an error occurs during this process, or the contents of the edit buffer are replaced (for example, by the *ex* **:edit** command) an error message shall be written and no more commands resulting from the execution of this command shall be processed.

Multiple *ex* commands can be specified by entering multiple commands on a single line using a vertical line to delimit them, or one per line, by escaping each <newline> with a backslash.

If no commands are specified:

1. If in *ex* command mode, it shall be as if the **print** command were specified.
2. Otherwise, no command shall be executed.

For the **append**, **change**, and **insert** commands, the input text shall be included as part of the command, and the terminating period can be omitted if the command ends the list of commands. The **open** and **visual** commands can be specified as one of the commands, in which case each marked line shall cause the editor to enter open or visual mode. If open or visual mode is exited using the *vi* **Q** command, the current line shall be set to the next marked line, and open or visual mode reentered, until the list of marked lines is exhausted.

The **global**, **v**, and **undo** commands cannot be used in *commands*. Marked lines may be deleted by commands executed for lines occurring earlier in the file than the marked lines. In this case, no commands shall be executed for the deleted lines.

If the remembered search direction is not set, the **global** and **v** commands shall set it to forward.

The **autoprint** and **autoindent** edit options shall be inhibited for the duration of the **g** or **v** command.

Current line: If no commands executed, set to the last marked line. Otherwise, as specified for the executed *ex* commands.

Current column: If no commands are executed, set to non-<blank>; otherwise, as specified for the individual *ex* commands.

Insert

Synopsis: **[***1addr***] i[**nsert**][!]**

Enter *ex* text input mode; the input text shall be placed before the specified line. If the line is zero or 1, the text shall be placed at the beginning of the edit buffer.

This command shall be affected by the **number** and **autoindent** edit options; following the command name with '!' shall cause the **autoindent** edit option setting to be toggled for the duration of this command only.

Current line: Set to the last input line; if no lines were input, set to the line before the specified line, or to the first line of the edit buffer if there are no lines preceding the specified line, or zero if the edit buffer is empty.

Current column: Set to non-<blank>.

Join

Synopsis: **[***2addr***] j[**oin**][!][***count***][***flags***]**

If *count* is specified:

If no address was specified, the **join** command shall behave as if *2addr* were the current line and the current line plus *count* (., . + *count*).

If one address was specified, the **join** command shall behave as if *2addr* were the specified address and the specified address plus *count* (*addr*,*addr* + *count*).

If two addresses were specified, the **join** command shall behave as if an additional address, equal to the last address plus *count* –1 (*addr1*,*addr2*,*addr2* + *count* –1), was specified.

If this would result in a second address greater than the last line of the edit buffer, it shall be corrected to be equal to the last line of the edit buffer.

If no *count* is specified:

If no address was specified, the **join** command shall behave as if *2addr* were the current line and the next line (., . +1).

If one address was specified, the **join** command shall behave as if *2addr* were the specified address and the next line (*addr*,*addr* +1).

Join the text from the specified lines together into a single line, which shall replace the specified lines.

If a `'!'` character is appended to the command name, the **join** shall be without modification of any line, independent of the current locale.

Otherwise, in the POSIX locale, set the current line to the first of the specified lines, and then, for each subsequent line, proceed as follows:

1. Discard leading <space>s from the line to be joined.
2. If the line to be joined is now empty, delete it, and skip steps 3 through 5.
3. If the current line ends in a <blank>, or the first character of the line to be joined is a `')'` character, join the lines without further modification.
4. If the last character of the current line is a `'.'`, join the lines with two <space>s between them.
5. Otherwise, join the lines with a single <space> between them.

Current line: Set to the first line specified.

Current column: Set to non-<blank>.

List

Synopsis: **[***2addr***] l[ist][***count***][***flags***]**

This command shall be equivalent to the *ex* command:

[*2addr***] p[rint][***count***] l[***flags***]**

See **Print** (on page 2580).

14411 **Map**

14412 *Synopsis*: `map[!][lhs rhs]`

14413 If *lhs* and *rhs* are not specified:

14414 1. If `'!'` is specified, write the current list of text input mode maps.

14415 2. Otherwise, write the current list of command mode maps.

14416 3. Do nothing more.

14417 Implementations may restrict the set of characters accepted in *lhs* or *rhs*, except that printable
14418 characters and <blank>s shall not be restricted. Additional restrictions shall be implementation-
14419 defined. In both *lhs* and *rhs*, any character can be escaped with a <control>-V, in which case the
14420 character shall not be used to delimit *lhs* from *rhs*, and the escaping <control>-V shall be
14421 discarded.

14422 If the character `'!'` is appended to the **map** command name, the mapping shall be effective
14423 during open or visual text input mode rather than **open** or **visual** command mode. This allows
14424 *lhs* to have two different **map** definitions at the same time: one for command mode and one for
14425 text input mode.

14426 For command mode mappings:

14427 When the *lhs* is entered as any part of a *vi* command in open or visual mode (but not as part
14428 of the arguments to the command), the action shall be as if the corresponding *rhs* had been
14429 entered.

14430 If any character in the command, other than the first, is escaped using a <control>-V
14431 character, that character shall not be part of a match to an *lhs*.

14432 It is unspecified whether implementations shall support **map** commands where the *lhs* is
14433 more than a single character in length, where the first character of the *lhs* is printable.

14434 If *lhs* contains more than one character and the first character is `'#'`, followed by a sequence
14435 of digits corresponding to a numbered function key, then when this function key is typed it
14436 shall be mapped to *rhs*. Characters other than digits following a `'#'` character also
14437 represent the function key named by the characters in the *lhs* following the `'#'` and may be
14438 mapped to *rhs*. It is unspecified how function keys are named or what function keys are
14439 supported.

14440 For text input mode mappings:

14441 When the *lhs* is entered as any part of text entered in open or visual text input modes, the
14442 action shall be as if the corresponding *rhs* had been entered.

14443 If any character in the input text is escaped using a <control>-V character, that character shall
14444 not be part of a match to an *lhs*.

14445 It is unspecified whether the *lhs* argument entered for the **map** or **unmap** commands is
14446 replaced in this fashion. Regardless of whether or not the replacement occurs, the effect of
14447 the command shall be as if the replacement had not occurred.

14448 If only part of the *lhs* is entered, it is unspecified how long the editor will wait for additional,
14449 possibly matching characters before treating the already entered characters as not matching the
14450 *lhs*.

14451 The *rhs* characters shall themselves be subject to remapping, unless otherwise specified by the
14452 **remap** edit option, except that if the characters in *lhs* occur as prefix characters in *rhs*, those
14453 characters shall not be remapped.

On 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 as if it occurred immediately.

Current line: Unchanged.

Current column: Unchanged.

Mark

Synopsis: `[`*1addr*`]` `ma[rk]` *character*
`[`*1addr*`]` `k` *character*

Implementations shall support *character* values of a single lowercase letter of the POSIX locale and the characters `'''` and `'''`; support of other characters is implementation-defined.

If executing the *vi* **m** command, set the specified mark to the current line and 1-based numbered character referenced by the current column, if any; otherwise, column position 1.

Otherwise, set the specified mark to the specified line and 1-based numbered first non-<blank> non- <newline> in the line, if any; otherwise, the last non-<newline> in the line, if any; otherwise, column position 1.

The mark shall remain associated with the line until the mark is reset or the line is deleted. If a deleted line is restored by a subsequent **undo** command, any marks previously associated with the line, which have not been reset, shall be restored as well. Any use of a mark not associated with a current line in the edit buffer shall be an error.

The marks **‘** and **’** shall be set as described previously, immediately before the following events occur in the editor:

1. The use of `'$'` as an *ex* address
2. The use of a positive decimal number as an *ex* address
3. The use of a search command as an *ex* address
4. The use of a mark reference as an *ex* address
5. The use of the following open and visual mode commands: <control>-], **%**, **(**, **)**, **[**, **]**, **{**, **}**.
6. The use of the following open and visual mode commands: **’**, **G**, **H**, **L**, **M**, **z** if the current line will change as a result of the command
7. The use of the open and visual mode commands: **/**, **?**, **N**, **‘**, **n** if the current line or column will change as a result of the command
8. The use of the *ex* mode commands: **z**, **undo**, **global**, **v**

For rules 1., 2., 3., and 4., the **‘** and **’** marks shall not be set if the *ex* command is parsed as specified by rule 6.a. in **Command Line Parsing in ex** (on page 2563).

For rules 5., 6., and 7., the **‘** and **’** marks shall not be set if the commands are used as motion commands in open and visual mode.

For rules 1., 2., 3., 4., 5., 6., 7., and 8., the **‘** and **’** marks shall not be set if the command fails.

The **‘** and **’** marks shall be set as described previously, each time the contents of the edit buffer are replaced (including the editing of the initial buffer), if in open or visual mode, or if in **ex** mode and the edit buffer is not empty, before any commands or movements (including commands or movements specified by the **–c** or **–t** options or the **+***command* argument) are executed on the edit buffer. If in open or visual mode, the marks shall be set as if executing the *vi*

m command; otherwise, as if executing the *ex* **mark** command.

When changing from **ex** mode to open or visual mode, if the ‘ and ’ marks are not already set, the ‘ and ’ marks shall be set as described previously.

Current line: Unchanged.

Current column: Unchanged.

Move

Synopsis: `[`*2addr*`] m[ove] `*1addr*` [`*flags*`]`

Move the specified lines after the specified destination line. A destination of line zero specifies that the lines shall be placed at the beginning of the edit buffer. It shall be an error if the destination line is within the range of lines to be moved.

Current line: Set to the last of the moved lines.

Current column: Set to non-<blank>.

Next

Synopsis: `n[ext][!][+`*command*`][`*file ...*`]`

If no ’!’ is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error, unless the file is successfully written as specified by the **autowrite** option.

If one or more files is specified:

1. Set the argument list to the specified filenames.
2. Set the current argument list reference to be the first entry in the argument list.
3. Set the current pathname to the first filename specified.

Otherwise:

1. It shall be an error if there are no more filenames in the argument list after the filename currently referenced.
2. Set the current pathname and the current argument list reference to the filename after the filename currently referenced in the argument list.

Replace the contents of the edit buffer with the contents of the file named by the current pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be empty.

This command shall be affected by the **autowrite** and **writeany** edit options.

The **+***command* option shall be <blank>-delimited; <blank>s can be escaped by preceding them with a backslash character. The **+***command* shall be interpreted as an *ex* command immediately after the contents of the edit buffer have been replaced and the current line and column have been set.

Current line: Set as described for the **edit** command.

Current column: Set as described for the **edit** command.

14531 **Number**

14532 *Synopsis*: **[***2addr***]** nu**[**mber**][***count***][***flags***]**
14533 **[***2addr***]** #**[***count***][***flags***]**

14534 These commands shall be equivalent to the *ex* command:

14535 **[***2addr***]** p**[**rint**][***count***]** #**[***flags***]**

14536 See **Print**.

14537 **Open**

14538 *Synopsis*: **[***1addr***]** o**[**pen**]** */pattern/* **[***flags***]**

14539 This command need not be supported on block-mode terminals or terminals with insufficient
14540 capabilities. If standard input, standard output, or standard error are not terminal devices, the
14541 results are unspecified.

14542 Enter open mode.

14543 The trailing delimiter can be omitted from pattern at the end of the command line. If pattern is
14544 empty (for example, "//") or not specified, the last regular expression used in the editor shall be
14545 used as the pattern. The pattern can be delimited by slashes (shown in the Synopsis), as well as
14546 any alphanumeric, or non-<blank> other than backslash, vertical line, double quote, or
14547 <newline>.

14548 *Current line*: Set to the specified line.

14549 *Current column*: Set to non-<blank>.

14550 **Preserve**

14551 *Synopsis*: pre**[**serve**]**

14552 Save the edit buffer in a form that can later be recovered by using the **–r** option or by using the *ex*
14553 **recover** command. After the file has been preserved, a mail message shall be sent to the user.
14554 This message shall be readable by invoking the *mailx* utility. The message shall contain the name
14555 of the file, the time of preservation, and an *ex* command that could be used to recover the file.
14556 Additional information may be included in the mail message.

14557 *Current line*: Unchanged.

14558 *Current column*: Unchanged.

14559 **Print**

14560 *Synopsis*: **[***2addr***]** p**[**rint**][***count***][***flags***]**

14561 Write the addressed lines. The behavior is unspecified if the number of columns on the display is
14562 less than the number of columns required to write any single character in the lines being written.

14563 Non-printable characters, except for the <tab>, shall be written as implementation-defined
14564 multi-character sequences.

14565 If the **#** flag is specified or the **number** edit option is set, each line shall be preceded by its line
14566 number in the following format:

14567 "%6dΔΔ", *<line number>*

14568 If the **l** flag is specified or the **list** edit option is set:

14569 1. The characters listed in the Base Definitions volume of IEEE Std 1003.1-200x, Table 5-1,
14570 Escape Sequences and Associated Actions shall be written as the corresponding escape
14571 sequence.

14572 2. Non-printable characters not in the Base Definitions volume of IEEE Std 1003.1-200x, Table
14573 5-1, Escape Sequences and Associated Actions shall be written as one three-digit octal
14574 number (with a preceding backslash) for each byte in the character (most significant byte
14575 first). If the size of a byte on the system is greater than 9 bits, the format used for non-
14576 printable characters is implementation-defined.

14577 3. The end of each line shall be marked with a `'$'`, and literal `'$'` characters within the line
14578 shall be written with a preceding backslash.

14579 Long lines shall be folded; the length at which folding occurs is unspecified, but should be
14580 appropriate for the output terminal, considering the number of columns of the terminal.

14581 If a line is folded, and the **l** flag is not specified and the **list** edit option is not set, it is unspecified
14582 whether a multi-column character at the folding position is separated; it shall not be discarded.

14583 *Current line*: Set to the last written line.

14584 *Current column*: Unchanged if the current line is unchanged; otherwise, set to non-<blank>.

14585 **Put**

14586 *Synopsis*: **[***1addr***]** pu**[**t**][***buffer***]**

14587 Append text from the specified buffer (by default, the unnamed buffer) to the specified line; line
14588 zero specifies that the text shall be placed at the beginning of the edit buffer. Each portion of a
14589 line in the buffer shall become a new line in the edit buffer, regardless of the mode of the buffer.

14590 *Current line*: Set to the last line entered into the edit buffer.

14591 *Current column*: Set to non-<blank>.

14592 **Quit**

14593 *Synopsis*: q**[**uit**][**!**]**

14594 If no `'!'` is appended to the command name:

14595 1. If the edit buffer has been modified since the last complete write, it shall be an error.

14596 2. If there are filenames in the argument list after the filename currently referenced, and the
14597 last command was not a **quit**, **wq**, **xit**, or **ZZ** (see **Exit** (on page 3220)) command, it shall be
14598 an error.

14599 Otherwise, terminate the editing session.

14600 **Read**

14601 *Synopsis*: **[***1addr***]** r**[**ead**][**!**][***file***]**

14602 If `'!'` is not the first non-<blank> to follow the command name, a copy of the specified file shall
14603 be appended into the edit buffer after the specified line; line zero specifies that the copy shall be
14604 placed at the beginning of the edit buffer. The number of lines and bytes read shall be written. If
14605 no *file* is named, the current pathname shall be the default. If there is no current pathname, then
14606 *file* shall become the current pathname. If there is no current pathname or *file* operand, it shall be
14607 an error. Specifying a *file* that is not of type regular shall have unspecified results.

Otherwise, if *file* is preceded by `'!'`, the rest of the line after the `'!'` shall have `'%'`, `'#'`, and `'!'` characters expanded as described in **Command Line Parsing in ex** (on page 2563).

The *ex* utility shall then pass two arguments to the program named by the *shell* edit option; the first shall be **–c** and the second shall be the expanded arguments to the **read** command as a single argument. The standard input of the program shall be set to the standard input of the *ex* program when it was invoked. The standard error and standard output of the program shall be appended into the edit buffer after the specified line.

Each line in the copied file or program output (as delimited by <newline>s or the end of the file or output if it is not immediately preceded by a <newline>), shall be a separate line in the edit buffer. Any occurrences of <carriage-return> and <newline> pairs in the output shall be treated as single <newline>s.

The special meaning of the `'!'` following the **read** command can be overridden by escaping it with a backslash character.

Current line: If no lines are added to the edit buffer, unchanged. Otherwise, if in open or visual mode, set to the first line entered into the edit buffer. Otherwise, set to the last line entered into the edit buffer.

Current column: Set to non-<blank>.

Recover

Synopsis: `rec[over][!]` *`file`*

If no `'!'` is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error.

If no *file* operand is specified, then the current pathname shall be used. If there is no current pathname or *file* operand, it shall be an error.

If no recovery information has previously been saved about *file*, the **recover** command shall behave identically to the **edit** command, and an informational message to this effect shall be written.

Otherwise, set the current pathname to *file*, and replace the current contents of the edit buffer with the recovered contents of *file*. If there are multiple instances of the file to be recovered, the one most recently saved shall be recovered, and an informational message that there are previous versions of the file that can be recovered shall be written. The editor shall behave as if the contents of the edit buffer have already been modified.

Current file: Set as described for the **edit** command.

Current column: Set as described for the **edit** command.

Rewind

Synopsis: `rew[ind][!]`

If no `'!'` is appended to the command name, and the edit buffer has been modified since the last complete write, it shall be an error, unless the file is successfully written as specified by the **autowrite** option.

If the argument list is empty, it shall be an error.

The current argument list reference and the current pathname shall be set to the first filename in the argument list.

Replace the contents of the edit buffer with the contents of the file named by the current pathname. If for any reason the contents of the file cannot be accessed, the edit buffer shall be empty.

This command shall be affected by the **autowrite** and **writeany** edit options.

Current line: Set as described for the **edit** command.

Current column: Set as described for the **edit** command.

Set

Synopsis: `se[t][`*`option`*`[=[`*`value`*`]] ...][no`*`option ...`*`][`*`option`*`? ...][all]`

When no arguments are specified, write the value of the **term** edit option and those options whose values have been changed from the default settings; when the argument *all* is specified, write all of the option values.

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 **set** *option* to turn them on or **set no***option* to turn them off; string and numeric options can be assigned by the form **set** *option*=*value*. Any <blank>s in strings can be included as is by preceding each <blank> with an escaping backslash. More than one option can be set or listed by a single set command by specifying multiple arguments, each separated from the next by one or more <blank>s.

See **Edit Options in ex** (on page 2593) for details about specific options.

Current line: Unchanged.

Current column: Unchanged.

Shell

Synopsis: `sh[ell]`

Invoke the program named in the **shell** edit option with the single argument **–i** (interactive mode). Editing shall be resumed when the program exits.

Current line: Unchanged.

Current column: Unchanged.

Source

Synopsis: `so[urce] `*`file`*

Read and execute *ex* commands from *file*. Lines in the file that are blank lines shall be ignored.

Current line: As specified for the individual *ex* commands.

Current column: As specified for the individual *ex* commands.

Substitute

Synopsis: **[***2addr***]** s**[**ubstitute**][**/*pattern*/*repl*/**[***options***][***count***][***flags***]]**
[*2addr***]** &**[***options***][***count***][***flags***]]**
[*2addr***]** ~**[***options***][***count***][***flags***]]**

Replace the first instance of the pattern *pattern* by the string *repl* on each specified line. (See **Regular Expressions in ex** (on page 2592) and **Replacement Strings in ex** (on page 2592).) Any non-alphabetic, non-<blank> delimiter other than `'\\'`, `'|'`, double quote, or <newline> can be used instead of `'/'`. Backslash characters can be used to escape delimiters, backslash characters, and other special characters.

The trailing delimiter can be omitted from *pattern* or from *repl* at the end of the command line. If both *pattern* and *repl* are not specified or are empty (for example, `"//"`), the last **s** command shall be repeated. If only *pattern* is not specified or is empty, the last regular expression used in the editor shall be used as the pattern. If only *repl* is not specified or is empty, the pattern shall be replaced by nothing. If the entire replacement pattern is `'%'`, the last replacement pattern to an **s** command shall be used.

Entering a <carriage-return> in *repl* (which requires an escaping backslash in *ex* mode and an escaping <control>-V in open or *vi* mode) shall split the line at that point, creating a new line in the edit buffer. The <carriage-return> shall be discarded.

If options include the letter `'g'` (**global**), all non-overlapping instances of the pattern in the line shall be replaced.

If options includes the letter `'c'` (**confirm**), then before each substitution the line shall be written; the written line shall reflect all previous substitutions. On the following line, <space>s shall be written beneath the characters from the line that are before the *pattern* to be replaced, and `'^'` characters written beneath the characters included in the *pattern* to be replaced. The *ex* utility shall then wait for a response from the user. An affirmative response shall cause the substitution to be done, while any other input shall not make the substitution. An affirmative response shall consist of a line with the affirmative response (as defined by the current locale) at the beginning of the line. This line shall be subject to editing in the same way as the *ex* command line.

If interrupted (see the ASYNCHRONOUS EVENTS section), any modifications confirmed by the user shall be preserved in the edit buffer after the interrupt.

If the remembered search direction is not set, the **s** command shall set it to forward.

In the second Synopsis, the **&** command shall repeat the previous substitution, as if the **&** command were replaced by:

`s/`*pattern*`/`*repl*`/`

where *pattern* and *repl* are as specified in the previous **s**, **&**, or **~** command.

In the third Synopsis, the **~** command shall repeat the previous substitution, as if the `'~'` were replaced by:

`s/`*pattern*`/`*repl*`/`

where *pattern* shall be the last regular expression specified to the editor, and *repl* shall be from the previous substitution (including **&** and **~**) command.

These commands shall be affected by the *LC_MESSAGES* environment variable.

Current line: Set to the last line in which a substitution occurred, or, unchanged if no substitution occurred.

14726 *Current column*: Set to non-<blank>.

14727 **Suspend**

14728 *Synopsis*: `su[spend][!]`
14729 `st[op][!]`

14730 Allow control to return to the invoking process; *ex* shall suspend itself as if it had received the
14731 SIGTSTP signal. The suspension shall occur only if job control is enabled in the invoking shell
14732 (see the description of *set* –**m**).

14733 These commands shall be affected by the **autowrite** and **writeany** edit options.

14734 The current **susp** character (see *stty*) shall be equivalent to the **suspend** command.

14735 **Tag**

14736 *Synopsis*: `ta[g][!]` *`tagstring`*

14737 The results are unspecified if the format of a tags file is not as specified by the *ctags* utility (see
14738 *ctags*) description.

14739 The **tag** command shall search for *tagstring* in the tag files referred to by the **tag** edit option, in
14740 the order they are specified, until a reference to *tagstring* is found. Files shall be searched from
14741 beginning to end. If no reference is found, it shall be an error and an error message to this effect
14742 shall be written. If the reference is not found, or if an error occurs while processing a file referred
14743 to in the **tag** edit option, it shall be an error, and an error message shall be written at the first
14744 occurrence of such an error.

14745 Otherwise, if the tags file contained a pattern, the pattern shall be treated as a regular expression
14746 used in the editor; for example, for the purposes of the **s** command.

14747 If the *tagstring* is in a file with a different name than the current pathname, set the current
14748 pathname to the name of that file, and replace the contents of the edit buffer with the contents of
14749 that file. In this case, if no `'!'` is appended to the command name, and the edit buffer has been
14750 modified since the last complete write, it shall be an error, unless the file is successfully written
14751 as specified by the **autowrite** option.

14752 This command shall be affected by the **autowrite**, **tag**, **taglength**, and **writeany** edit options.

14753 *Current line*: If the tags file contained a line number, set to that line number. If the line number is
14754 larger than the last line in the edit buffer, an error message shall be written and the current line
14755 shall be set as specified for the **edit** command.

14756 If the tags file contained a pattern, set to the first occurrence of the pattern in the file. If no
14757 matching pattern is found, an error message shall be written and the current line shall be set as
14758 specified for the **edit** command.

14759 *Current column*: If the tags file contained a line-number reference and that line-number was not
14760 larger than the last line in the edit buffer, or if the tags file contained a pattern and that pattern
14761 was found, set to non-<blank>. Otherwise, set as specified for the **edit** command.

Unabbreviate

Synopsis: `una[bbrev]` *lhs*

If *lhs* is not an entry in the current list of abbreviations (see **Abbreviate** (on page 2571)), it shall be an error. Otherwise, delete *lhs* from the list of abbreviations.

Current line: Unchanged.

Current column: Unchanged.

Undo

Synopsis: `u[ndo]`

Reverse the changes made by the last command that modified the contents of the edit buffer, including **undo**. For this purpose, the **global**, **v**, **open**, and **visual** commands, and commands resulting from buffer executions and mapped character expansions, are considered single commands.

If no action that can be undone preceded the **undo** command, it shall be an error.

If the **undo** command restores lines that were marked, the mark shall also be restored unless it was reset subsequent to the deletion of the lines.

Current line:

1. If lines are added or changed in the file, set to the first line added or changed.
2. Set to the line before the first line deleted, if it exists.
3. Set to 1 if the edit buffer is not empty.
4. Set to zero.

Current column: Set to non-<blank>.

Unmap

Synopsis: `unm[ap][!]` *lhs*

If '`!`' is appended to the command name, and if *lhs* is not an entry in the list of text input mode map definitions, it shall be an error. Otherwise, delete *lhs* from the list of text input mode map definitions.

If no '`!`' is appended to the command name, and if *lhs* is not an entry in the list of command mode map definitions, it shall be an error. Otherwise, delete *lhs* from the list of command mode map definitions.

Current line: Unchanged.

Current column: Unchanged.

Version

Synopsis: `ve[rsion]`

Write a message containing version information for the editor. The format of the message is unspecified.

Current line: Unchanged.

Current column: Unchanged.

14799 **Visual**

14800 *Synopsis*: **[***1addr***]** vi**[**sual**][***type***][***count***][***flags***]**

14801 If *ex* is currently in open or visual mode, the Synopsis and behavior of the visual command shall
14802 be the same as the **edit** command, as specified by **Edit** (on page 2573).

14803 Otherwise, this command need not be supported on block-mode terminals or terminals with
14804 insufficient capabilities. If standard input, standard output, or standard error are not terminal
14805 devices, the results are unspecified.

14806 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in
14807 **window** (on page 2599)). If the **'^'** type character was also specified, the **window** edit option
14808 shall be set before being used by the type character.

14809 Enter visual mode. If *type* is not specified, it shall be as if a *type* of **'+'** was specified. The *type*
14810 shall cause the following effects:

14811 + Place the beginning of the specified line at the top of the display.

14812 - Place the end of the specified line at the bottom of the display.

14813 . Place the beginning of the specified line in the middle of the display.

14814 ^ If the specified line is less than or equal to the value of the **window** edit option, set the line
14815 to 1; otherwise, decrement the line by the value of the **window** edit option minus 1. Place
14816 the beginning of this line as close to the bottom of the displayed lines as possible, while still
14817 displaying the value of the **window** edit option number of lines.

14818 *Current line*: Set to the specified line.

14819 *Current column*: Set to non-<blank>.

14820 **Write**

14821 *Synopsis*: **[***2addr***]** w**[**rite**][**!**][**>>**][***file***]**
14822 **[***2addr***]** w**[**rite**][**!**][***file***]**
14823 **[***2addr***]** wq**[**!**][**>>**][***file***]**

14824 If no lines are specified, the lines shall default to the entire file.

14825 The command **wq** shall be equivalent to a **write** command followed by a **quit** command; **wq!**
14826 shall be equivalent to **write!** followed by **quit**. In both cases, if the **write** command fails, the
14827 **quit** shall not be attempted.

14828 If the command name is not followed by one or more <blank>s, or *file* is not preceded by a **'!'**
14829 character, the **write** shall be to a file.

14830 1. If the **>>** argument is specified, and the file already exists, the lines shall be appended to
14831 the file instead of replacing its contents. If the **>>** argument is specified, and the file does
14832 not already exist, it is unspecified whether the write shall proceed as if the **>>** argument
14833 had not been specified or if the write shall fail.

14834 2. If the **readonly** edit option is set (see **readonly** (on page 2596)), the **write** shall fail.

14835 3. If *file* is specified, and is not the current pathname, and the file exists, the **write** shall fail.

14836 4. If *file* is not specified, the current pathname shall be used. If there is no current pathname,
14837 the **write** command shall fail.

14838 5. If the current pathname is used, and the current pathname has been changed by the **file** or
14839 **read** commands, and the file exists, the **write** shall fail. If the **write** is successful,

subsequent **write**s shall not fail for this reason (unless the current pathname is changed again).

6. If the whole edit buffer is not being written, and the file to be written exists, the **write** shall fail.

For rules 1., 2., 4., and 5., the **write** can be forced by appending the character '!' to the command name.

For rules 2., 4., and 5., the **write** can be forced by setting the **writeany** edit option.

Additional, implementation-defined tests may cause the **write** to fail.

If the edit buffer is empty, a file without any contents shall be written.

An informational message shall be written noting the number of lines and bytes written.

Otherwise, if the command is followed by one or more <blank>s, and the file is preceded by '!', the rest of the line after the '!' shall have '%', '#', and '!' characters expanded as described in **Command Line Parsing in ex** (on page 2563).

The *ex* utility shall then pass two arguments to the program named by the **shell** edit option; the first shall be **–c** and the second shall be the expanded arguments to the **write** command as a single argument. The specified lines shall be written to the standard input of the command. The standard error and standard output of the program. if any, shall be written as described for the **print** command. If the last character in that output is not a <newline>, a <newline> shall be written at the end of the output.

The special meaning of the '!' following the **write** command can be overridden by escaping it with a backslash character.

Current line: Unchanged.

Current column: Unchanged.

Write and Exit

Synopsis: **[***2addr***]** x**[**it**][**!**][***file***]**

If the edit buffer has not been modified since the last complete **write**, **xit** shall be equivalent to the **quit** command, or if a '!' is appended to the command name, to **quit!**.

Otherwise, **xit** shall be equivalent to the **wq** command, or if a '!' is appended to the command name, to **wq!**.

Current line: Unchanged.

Current column: Unchanged.

Yank

Synopsis: **[***2addr***]** ya**[**nk**][***buffer***][***count***]**

Copy the specified lines to the specified buffer (by default, the unnamed buffer), which shall become a line-mode buffer.

Current line: Unchanged.

Current column: Unchanged.

Adjust Window

Synopsis: **[***1addr***] z[!][***type ...***][***count***][***flags***]**

If no line is specified, the current line shall be the default; if *type* is omitted as well, the current line value shall first be incremented by 1. If incrementing the current line would cause it to be greater than the last line in the edit buffer, it shall be an error.

If there are <blank>s between the *type* argument and the preceding **z** command name or optional '!' character, it shall be an error.

If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in **window** (on page 2599)). If *count* is omitted, it shall default to 2 times the value of the **scroll** edit option, or if ! was specified, the number of lines in the display minus 1.

If *type* is omitted, then *count* lines starting with the specified line shall be written. Otherwise, *count* lines starting with the line specified by the *type* argument shall be written.

The *type* argument shall change the lines to be written. The possible values of *type* are as follows:

– The specified line shall be decremented by the following value:

(((number of ''–'' characters) x *count*) –1)

If the calculation would result in a number less than 1, it shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit buffer has been written.

\+ The specified line shall be incremented by the following value:

(((number of ''+'' characters) –1) x *count*) +1

If the calculation would result in a number greater than the last line in the edit buffer, it shall be an error. Write lines from the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit buffer has been written.

=,. If more than a single '.' or '=' is specified, it shall be an error. The following steps shall be taken:

1. If *count* is zero, nothing shall be written.
2. Write as many of the *N* lines before the current line in the edit buffer as exist. If *count* or '!' was specified, *N* shall be:

   (*count* –1) /2

   Otherwise, *N* shall be:

   (*count* –3) /2

   If *N* is a number less than 3, no lines shall be written.
3. If '=' was specified as the type character, write a line consisting of the smaller of the number of columns in the display divided by two, or 40 '–' characters.
4. Write the current line.
5. Repeat step 3.
6. Write as many of the *N* lines after the current line in the edit buffer as exist. *N* shall be defined as in step 2. If *N* is a number less than 3, no lines shall be written. current line in the edit buffer as exist. If count is less than 3, no lines shall be written.

14916 ^ The specified line shall be decremented by the following value:

14917 (((number of ''^'' characters) +1) x *count*) −1

14918 If the calculation would result in a number less than 1, it shall be an error. Write lines from
14919 the edit buffer, starting at the new value of line, until *count* lines or the last line in the edit
14920 buffer has been written.

14921 *Current line*: Set to the last line written, unless the type is **=**, in which case, set to the specified
14922 line.

14923 *Current column*: Set to non-<blank>.

14924 **Escape**

14925 *Synopsis*: ! *command*
14926 [*addr*]! *command*

14927 The contents of the line after the '!' shall have '%', '#', and '!' characters expanded as
14928 described in **Command Line Parsing in ex** (on page 2563). If the expansion causes the text of the
14929 line to change, it shall be redisplayed, preceded by a single '!' character.

14930 The *ex* utility shall execute the program named by the **shell** edit option. It shall pass two
14931 arguments to the program; the first shall be **−c**, and the second shall be the expanded arguments
14932 to the **!** command as a single argument.

14933 If no lines are specified, the standard input, standard output, and standard error of the program
14934 shall be set to the standard input, standard output, and standard error of the *ex* program when it
14935 was invoked. In addition, a warning message shall be written if the edit buffer has been
14936 modified since the last complete write, and the **warn** edit option is set.

14937 If lines are specified, they shall be passed to the program as standard input, and the standard
14938 output and standard error of the program shall replace those lines in the edit buffer. Each line in
14939 the program output (as delimited by <newline>s or the end of the output if it is not immediately
14940 preceded by a <newline>), shall be a separate line in the edit buffer. Any occurrences of
14941 <carriage-return> and <newline> pairs in the output shall be treated as single <newline>s. The
14942 specified lines shall be copied into the unnamed buffer before they are replaced, and the
14943 unnamed buffer shall become a line-mode buffer.

14944 If in *ex* mode, a single '!' character shall be written when the program completes.

14945 This command shall be affected by the **shell** and **warn** edit options. If no lines are specified, this
14946 command shall be affected by the **autowrite** and **writeany** edit options. If lines are specified, this
14947 command shall be affected by the **autoprint** edit option.

14948 *Current line*:

14949 1. If no lines are specified, unchanged.
14950 2. Otherwise, set to the last line read in, if any lines are read in.
14951 3. Otherwise, set to the line before the first line of the lines specified, if that line exists.
14952 4. Otherwise, set to the first line of the edit buffer if the edit buffer is not empty.
14953 5. Otherwise, set to zero.

14954 *Current column*: If no lines are specified, unchanged. Otherwise, set to non-<blank>.

14955 **Shift Left**

14956 *Synopsis*: **[***2addr***] <[<** ...**][***count***][***flags***]**

14957 Shift the specified lines to the start of the line; the number of column positions to be shifted shall
14958 be the number of command characters times the value of the **shiftwidth** edit option. Only
14959 leading <blank>s shall be deleted or changed into other <blank>s in shifting; other characters
14960 shall not be affected.

14961 Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode
14962 buffer.

14963 This command shall be affected by the **autoprint** edit option.

14964 *Current line*: Set to the last line in the lines specified.

14965 *Current column*: Set to non-<blank>.

14966 **Shift Right**

14967 *Synopsis*: **[***2addr***] >[>** ...**][***count***][***flags***]**

14968 Shift the specified lines away from the start of the line; the number of column positions to be
14969 shifted shall be the number of command characters times the value of the **shiftwidth** edit option.
14970 The shift shall be accomplished by adding <blank>s as a prefix to the line or changing leading
14971 <blank>s into other <blank>s. Empty lines shall not be changed.

14972 Lines to be shifted shall be copied into the unnamed buffer, which shall become a line-mode
14973 buffer.

14974 This command shall be affected by the **autoprint** edit option.

14975 *Current line*: Set to the last line in the lines specified.

14976 *Current column*: Set to non-<blank>.

14977 **<control>-D**

14978 *Synopsis*: **<control>-D**

14979 Write the next *n* lines, where *n* is the minimum of the values of the **scroll** edit option and the
14980 number of lines after the current line in the edit buffer. If the current line is the last line of the
14981 edit buffer it shall be an error.

14982 *Current line*: Set to the last line written.

14983 *Current column*: Set to non-<blank>.

14984 **Write Line Number**

14985 *Synopsis*: **[***1addr***] = [***flags***]**

14986 If *line* is not specified, it shall default to the last line in the edit buffer. Write the line number of
14987 the specified line.

14988 *Current line*: Unchanged.

14989 *Current column*: Unchanged.

14990 **Execute**

14991 *Synopsis*: **[***2addr***]** **@** *buffer*
14992 **[***2addr***]** ***** *buffer*

14993 If no buffer is specified or is specified as '@' or '*', the last buffer executed shall be used. If no
14994 previous buffer has been executed, it shall be an error.

14995 For each line specified by the addresses, set the current line ('.') to the specified line, and
14996 execute the contents of the named *buffer* (as they were at the time the **@** command was executed)
14997 as *ex* commands. For each line of a line-mode buffer, and all but the last line of a character-mode
14998 buffer, the *ex* command parser shall behave as if the line was terminated by a <newline>.

14999 If an error occurs during this process, or a line specified by the addresses does not exist when the
15000 current line would be set to it, or more than a single line was specified by the addresses, and the
15001 contents of the edit buffer are replaced (for example, by the *ex* **:edit** command) an error message
15002 shall be written, and no more commands resulting from the execution of this command shall be
15003 processed.

15004 *Current line*: As specified for the individual *ex* commands.

15005 *Current column*: As specified for the individual *ex* commands.

15006 **Regular Expressions in ex**

15007 The *ex* utility shall support regular expressions that are a superset of the basic regular
15008 expressions described in the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.3, Basic
15009 Regular Expressions. A null regular expression ("//") shall be equivalent to the last regular
15010 expression encountered.

15011 Regular expressions can be used in addresses to specify lines and, in some commands (for
15012 example, the **substitute** command), to specify portions of a line to be substituted.

15013 The following constructs can be used to enhance the basic regular expressions:

15014 \< Match the beginning of a *word*. (See the definition of *word* at the beginning of **Command**
15015 **Descriptions in ex** (on page 2569).)

15016 \> Match the end of a *word*.

15017 ~ Match the replacement part of the last **substitute** command. The tilde ('~') character can
15018 be escaped in a regular expression to become a normal character with no special meaning.
15019 The backslash shall be discarded.

15020 When the editor option **magic** is not set, the only characters with special meanings shall be '^'
15021 at the beginning of a pattern, '$' at the end of a pattern, and '\'. The characters '.', '*',
15022 '[', and '~' shall be treated as ordinary characters unless preceded by a '\'; when preceded
15023 by a '\' they shall regain their special meaning, or in the case of backslash, be handled as a
15024 single backslash. Backslashes used to escape other characters shall be discarded.

15025 **Replacement Strings in ex**

15026 The character '&' ('\&' if the editor option **magic** is not set) in the replacement string shall
15027 stand for the text matched by the pattern to be replaced. The character '~' ('\~' if **magic** is not
15028 set) shall be replaced by the replacement part of the previous **substitute** command. The
15029 sequence '\n', where *n* is an integer, shall be replaced by the text matched by the pattern
15030 enclosed in the *n*th set of parentheses '\(' and '\)'.

15031 The strings '\l', '\u', '\L', and '\U' can be used to modify the case of elements in the
15032 replacement string (using the '\&' or "\"digit) notation. The string '\l' ('\u') shall cause

the character that follows to be converted to lowercase (uppercase). The string `'\L'` (`'\U'`) shall cause all characters subsequent to it 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.

Otherwise, any character following a backslash shall be treated as that literal character, and the escaping backslash shall be discarded.

An example of case conversion with the **s** command is as follows:

```
: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.
```

Edit Options in ex

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.

autoindent, ai

[Default *unset*]

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 another line, as follows:

1. If in open or visual mode and the text input is part of a line-oriented command (see the EXTENDED DESCRIPTION in *vi*), align to the first column. Otherwise, if in open or visual mode, indentation for each line shall be set as follows:
   a. If a line was previously inserted as part of this command, it shall be set to the indentation of the last inserted line by default, or as otherwise specified for the <control>-D character in **Input Mode Commands in vi** (on page 3220).
   b. Otherwise, it shall be set to the indentation of the previous current line, if any; otherwise, to the first column.
2. For the *ex* **a**, **i**, and **c** commands, indentation for each line shall be set as follows:
   a. If a line was previously inserted as part of this command, it shall be set to the indentation of the last inserted line by default, or as otherwise specified for the *eof* character in **Scroll** (on page 2567).
   b. Otherwise, if the command is the *ex* **a** command, it shall be set to the line appended after, if any; otherwise to the first column.
   c. Otherwise, if the command is the *ex* **i** command, it shall be set to the line inserted before, if any; otherwise to the first column.
   d. Otherwise, if the command is the *ex* **c** command, it shall be set to the indentation of the line replaced.

15073 **autoprint, ap**

15074 [Default *set*]

15075 If **autoprint** is set, the current line shall be written after each *ex* command that modifies the
15076 contents of the current edit buffer, and after each **tag** command for which the tag search pattern
15077 was found or tag line number was valid, unless:

15078 1. The command was executed while in open or visual mode.

15079 2. The command was executed as part of a **global** or **v** command or **@** buffer execution.

15080 3. The command was the form of the **read** command that reads a file into the edit buffer.

15081 4. The command was the **append**, **change**, or **insert** command.

15082 5. The command was not terminated by a <newline>.

15083 6. The current line shall be written by a flag specified to the command; for example, **delete #**
15084 shall write the current line as specified for the flag modifier to the **delete** command, and
15085 not as specified by the **autoprint** edit option.

15086 **autowrite, aw**

15087 [Default *unset*]

15088 If **autowrite** is set, and the edit buffer has been modified since it was last completely written to
15089 any file, the contents of the edit buffer shall be written as if the *ex* **write** command had been
15090 specified without arguments, before each command affected by the **autowrite** edit option is
15091 executed. Appending the character '!' to the command name of any of the *ex* commands
15092 except '!' shall prevent the write. If the write fails, it shall be an error and the command shall
15093 not be executed.

15094 **beautify, bf**

15095 XSI [Default *unset*]

15096 If **beautify** is set, all non-printable characters, other than <tab>s, <newline>s, and <form-feed>s,
15097 shall be discarded from text read in from files.

15098 **directory, dir**

15099 [Default *implementation-defined*]

15100 The value of this option specifies the directory in which the editor buffer is to be placed. If this
15101 directory is not writable by the user, the editor shall quit.

15102 **edcompatible, ed**

15103 [Default *unset*]

15104 Causes the presence of **g** and **c** suffixes on substitute commands to be remembered, and toggled
15105 by repeating the suffixes.

errorbells, eb

[Default *unset*]

If the editor is in *ex* mode, and the terminal does not support a standout mode (such as inverse video), and **errorbells** is set, error messages shall be preceded by alerting the terminal.

exrc

[Default *unset*]

If **exrc** is set, *ex* shall access any **.exrc** file in the current directory, as described in **Initialization in ex and vi** (on page 2559). 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* environment variable.

ignorecase, ic

[Default *unset*]

If **ignorecase** is set, characters that have uppercase and lowercase representations shall have those representations considered as equivalent for purposes of regular expression comparison.

The **ignorecase** edit option shall affect all remembered regular expressions; for example, unsetting the **ignorecase** edit option shall cause a subsequent *vi* **n** command to search for the last basic regular expression in a case-sensitive fashion.

list

[Default *unset*]

If **list** is set, edit buffer lines written while in *ex* command mode shall be written as specified for the **print** command with the **l** flag specified. In open or visual mode, each edit buffer line shall be displayed as specified for the *ex* print command with the **l** flag specified. In open or visual text input mode, when the cursor does not rest on any character in the line, it shall rest on the `'$'` marking the end of the line.

magic

[Default *set*]

If **magic** is set, modify the interpretation of characters in regular expressions and substitution replacement strings (see **Regular Expressions in ex** (on page 2592) and **Replacement Strings in ex** (on page 2592)).

mesg

[Default *set*]

If **mesg** is set, the permission for others to use the **write** or **talk** commands to write to the terminal shall be turned on while in open or visual mode. The shell-level command *mesg* **n** shall take precedence over any setting of the *ex* **mesg** option; that is, if **mesg y** was issued before the editor started (or in a shell escape), such as:

```
:!mesg y
```

the **mesg** option in *ex* shall suppress incoming messages, but the **mesg** option shall not enable incoming messages if **mesg n** was issued.

number, nu

[Default *unset*]

If **number** is set, edit buffer lines written while in *ex* command mode shall be written with line numbers, in the format specified by the **print** command with the **#** flag specified. In *ex* text input mode, each line shall be preceded by the line number it will have in the file.

In open or visual mode, each edit buffer line shall be displayed with a preceding line number, in the format specified by the *ex* **print** command with the **#** flag specified. This line number shall not be considered part of the line for the purposes of evaluating the current column; that is, column position 1 shall be the first column position after the format specified by the **print** command.

paragraphs, para

[Default in the POSIX locale `IPLPPPQPP LIpplpipbp`]

The **paragraphs** edit option shall define additional paragraph boundaries for the open and visual mode commands. The **paragraphs** edit option can be set to a character string consisting of zero or more character pairs. It shall be an error to set it to an odd number of characters.

prompt

[Default *set*]

If **prompt** is set, *ex* command mode input shall be prompted for with a colon (`':'`); when unset, no prompt shall be written.

readonly

[Default *see text*]

If **readonly** edit option is set, read-only mode shall be enabled (see **Write** (on page 2587)). The **readonly** edit option shall be initialized to set if either of the following conditions are true:

- The command-line option –R was specified.
- Performing actions equivalent to the *access*( ) function called with the following arguments indicates that the file lacks write permission:
  1. The current pathname is used as the *path* argument.
  2. The constant **W_OK** is used as the *amode* argument.

The **readonly** edit option may be initialized to set for other, implementation-defined reasons. The **readonly** edit option shall not be initialized to unset based on any special privileges of the user or process. The **readonly** edit option shall be reinitialized each time that the contents of the edit buffer are replaced (for example, by an **edit** or **next** command) unless the user has explicitly set it, in which case it shall remain set until the user explicitly unsets it. Once unset, it shall again be reinitialized each time that the contents of the edit buffer are replaced.

redraw

[Default *unset*]

The editor simulates an intelligent terminal on a dumb terminal. (Since this is likely to require a large amount of output to the terminal, it is useful only at high transmission speeds.)

remap

[Default *set*]

If **remap** is set, map translation shall allow for maps defined in terms of other maps; translation shall continue until a final product is obtained. If unset, only a one-step translation shall be done.

report

[Default 5]

The value of this **report** edit option specifies what number of lines being added, copied, deleted, or modified in the edit buffer will cause an informational message to be written to the user. The following conditions shall cause an informational message. The message shall contain the number of lines added, copied, deleted, or modified, but is otherwise unspecified.

- An *ex* or *vi* editor command, other than **open**, **undo**, or **visual**, that modifies at least the value of the **report** edit option number of lines, and which is not part of an *ex* **global** or **v** command, or *ex* or *vi* buffer execution, shall cause an informational message to be written.
- An *ex* **yank** or *vi* **y** or **Y** command, that copies at least the value of the **report** edit option plus 1 number of lines, and which is not part of an *ex* **global** or **v** command, or *ex* or *vi* buffer execution, shall cause an informational message to be written.
- An *ex* **global**, **v**, **open**, **undo**, or **visual** command or *ex* or *vi* buffer execution, that adds or deletes a total of at least the value of the **report** edit option number of lines, and which is not part of an *ex* **global** or **v** command, or *ex* or *vi* buffer execution, shall cause an informational message to be written. (For example, if 3 lines were added and 8 lines deleted during an *ex* **visual** command, 5 would be the number compared against the **report** edit option after the command completed.

scroll, scr

[Default (number of lines in the display –1)/2]

The value of the **scroll** edit option shall determine the number of lines scrolled by the *ex* <control>-D and **z** commands. For the *vi* <control>-D and <control>-U commands, it shall be the initial number of lines to scroll when no previous <control>-D or <control>-U command has been executed.

sections

[Default in the POSIX locale `NHSHH HUnhsh`]

The **sections** edit option shall define additional section boundaries for the open and visual mode commands. The **sections** edit option can be set to a character string consisting of zero or more character pairs; it shall be an error to set it to an odd number of characters.

15215 **shell, sh**

15216 [Default from the environment variable *SHELL*]

15217 The value of this option shall be a string. The default shall be taken from the *SHELL*
15218 environment variable. If the *SHELL* environment variable is null or empty, the *sh* (see *sh*) utility
15219 shall be the default.

15220 **shiftwidth, sw**

15221 [Default 8]

15222 The value of this option shall give the width in columns of an indentation level used during
15223 autoindentation and by the shift commands (< and >).

15224 **showmatch, sm**

15225 [Default *unset*]

15226 The functionality described for the **showmatch** edit option need not be supported on block-
15227 mode terminals or terminals with insufficient capabilities.

15228 If **showmatch** is set, in open or visual mode, when a `')'` or `'}'` is typed, if the matching `'('` or
15229 `'{'` is currently visible on the display, the matching `'('` or `'{'` shall be flagged moving the
15230 cursor to its location for an unspecified amount of time.

15231 **showmode**

15232 [Default *unset*]

15233 If **showmode** is set, in open or visual mode, the current mode that the editor is in shall be
15234 displayed on the last line of the display. Command mode and text input mode shall be
15235 differentiated; other unspecified modes and implementation-defined information may be
15236 displayed.

15237 **slowopen**

15238 [Default *unset*]

15239 If **slowopen** is set during open and visual text input modes, the editor shall not update portions
15240 of the display other than those display line columns that display the characters entered by the
15241 user (see **Input Mode Commands in vi** (on page 3220)).

15242 **tabstop, ts**

15243 [Default 8]

15244 The value of this edit option shall specify the column boundary used by a <tab> in the display
15245 (see **autoprint, ap** (on page 2594) and **Input Mode Commands in vi** (on page 3220)).

15246 **taglength, tl**

15247 [Default zero]

15248 The value of this edit option shall specify the maximum number of characters that are
15249 considered significant in the user-specified tag name and in the tag name from the tags file. If the
15250 value is zero, all characters in both tag names shall be significant.

15251 **tags**

15252 [Default *see text*]

15253 The value of this edit option shall be a string of <blank>-delimited pathnames of files used by
15254 the **tag** command. The default value is unspecified.

15255 **term**

15256 [Default from the environment variable *TERM*]

15257 The value of this edit option shall be a string. The default shall be taken from the *TERM* variable
15258 in the environment. If the *TERM* environment variable is empty or null, the default is
15259 unspecified. The editor shall use the value of this edit option to determine the type of the display
15260 device.

15261 The results are unspecified if the user changes the value of the term edit option after editor
15262 initialization.

15263 **terse**

15264 [Default *unset*]

15265 If **terse** is set, error messages may be less verbose. However, except for this caveat, error
15266 messages are unspecified. Furthermore, not all error messages need change for different settings
15267 of this option.

15268 **warn**

15269 [Default *set*]

15270 If **warn** is set, and the contents of the edit buffer have been modified since they were last
15271 completely written, the editor shall write a warning message before certain **!** commands (see
15272 **Escape** (on page 2590)).

15273 **window**

15274 [Default *see text*]

15275 A value used in open and visual mode, by the <control>-B and <control>-F commands, and, in
15276 visual mode, to specify the number of lines displayed when the screen is repainted.

15277 If the **−w** command-line option is not specified, the default value shall be set to the value of the
15278 *LINES* environment variable. If the *LINES* environment variable is empty or null, the default
15279 shall be the number of lines in the display minus 1.

15280 Setting the **window** edit option to zero or to a value greater than the number of lines in the
15281 display minus 1 (either explicitly or based on the **−w** option or the *LINES* environment variable)
15282 shall cause the **window** edit option to be set to the number of lines in the display minus 1.

15283 The baud rate of the terminal line may change the default in an implementation-defined manner.

wrapmargin, wm

[Default 0]

If the value of this edit option is zero, it shall have no effect.

If not in the POSIX locale, the effect of this edit option is implementation-defined.

Otherwise, it shall specify a number of columns from the ending margin of the terminal.

During open and visual text input modes, for each character for which any part of the character is displayed in a column that is less than **wrapmargin** columns from the ending margin of the display line, the editor shall behave as follows:

1. If the character triggering this event is a <blank>, it, and all immediately preceding <blank>s on the current line entered during the execution of the current text input command, shall be discarded, and the editor shall behave as if the user had entered a single <newline> instead. In addition, if the next user-entered character is a <space>, it shall be discarded as well.

2. Otherwise, if there are one or more <blank>s on the current line immediately preceding the last group of inserted non-<blank>s which was entered during the execution of the current text input command, the <blank>s shall be replaced as if the user had entered a single <newline> instead.

If the **autoindent** edit option is set, and the events described in 1. or 2. are performed, any <blank>s at or after the cursor in the current line shall be discarded.

The ending margin shall be determined by the system or overridden by the user, as described for *COLUMNS* in in the ENVIRONMENT VARIABLES section and the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

wrapscan, ws

[Default *set*]

If **wrapscan** is set, searches (the *ex* **/** or **?** addresses, or open and visual mode **/**, **?**, **N**, and **n** commands) shall wrap around the beginning or end of the edit buffer; when unset, searches shall stop at the beginning or end of the edit buffer.

writeany, wa

[Default *unset*]

If **writeany** is set, some of the checks performed when executing the *ex* **write** commands shall be inhibited, as described in editor option **autowrite**.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

When any error is encountered and the standard input is not a terminal device file, *ex* shall not write the file or return to command or text input mode, and shall terminate with a non-zero exit status.

Otherwise, when an unrecoverable error is encountered, it shall be equivalent to a SIGHUP asynchronous event.

Otherwise, when an error is encountered, the editor shall behave as specified in **Command Line Parsing in ex** (on page 2563).

APPLICATION USAGE

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.

EXAMPLES

None.

RATIONALE

The *ex*/*vi* specification is based on the historical practice found in the 4 BSD and System V implementations of *ex* and *vi*. A freely redistributable implementation of *ex*/*vi*, which is tracking IEEE Std 1003.1-200x fairly closely, and demonstrates the intended changes between historical implementations and IEEE Std 1003.1-200x, may be obtained by anonymous FTP from:

```
ftp://ftp.rdg.opengroup/pub/mirrors/nvi
```

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 users might expect.

It is recognized that *ex* visual mode and related features would be difficult, if not impossible, to implement satisfactorily on a block-mode terminal, or a terminal without any form of cursor addressing; 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.

Options

The **–c** replacement for **+***command* was inspired by the **–e** option of *sed*. Historically, all such commands (see **edit** and **next** as well) were executed from the last line of the edit buffer. This meant, for example, that `"+/pattern"` would fail unless the **wrapscan** option was set. IEEE Std 1003.1-200x requires conformance to historical practice. Historically, some implementations restricted the *ex* commands that could be listed as part of the command line arguments. For consistency, IEEE Std 1003.1-200x does not permit these restrictions.

In historical implementations of the editor, the **–R** option (and the **readonly** edit option) only prevented overwriting of files; appending to files was still permitted, mapping loosely into the *csh* **noclobber** variable. Some implementations, however, have not followed this semantic, and **readonly** does not permit appending either. IEEE Std 1003.1-200x follows the latter practice, believing that it is a more obvious and intuitive meaning of **readonly**.

The **–s** option suppresses all interactive user feedback and is useful for editing scripts in batch jobs. The list of specific effects is historical practice. The terminal type ''incapable of supporting open and visual modes'' has historically been named ''dumb''.

The **–t** option was required because the *ctags* utility appears in IEEE Std 1003.1-200x and the option is available in all historical implementations of *ex*.

Historically, the *ex* and *vi* utilities accepted a **–x** option, which did encryption based on the algorithm found in the historical *crypt* utility. 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. In addition, it did not

historically provide the level of security that users might expect.

Standard Input

An end-of-file condition is not equivalent to an end-of-file character. A common end-of-file character, <control>-D, is historically an *ex* command.

There was no maximum line length in historical implementations of *ex*. Specifically, as it was parsed in chunks, the addresses had a different maximum length than the filenames. Further, the maximum line buffer size was declared as BUFSIZ, which was different lengths on different systems. This version selected the value of {LINE_MAX} to impose a reasonable restriction on portable usage of *ex* and to aid test suite writers in their development of realistic tests that exercise this limit.

Input Files

It was an explicit decision by the standard developers that a <newline> be added to any file lacking one. It was believed that this feature of *ex* and *vi* was relied on by users in order to make text files lacking a trailing <newline> more portable. It is recognized that this will require a user-specified option or extension for implementations that permit *ex* and *vi* to edit files of type other than text if such files are not otherwise identified by the system. It was agreed that the ability to edit files of arbitrary type can be useful, but it was not considered necessary to mandate that an *ex* or *vi* implementation be required to handle files other than text files.

The paragraph in the INPUT FILES section, ‘‘By default, . . .’’, is intended to close a long-standing security problem in *ex* and *vi*, that of the ‘‘modeline’’ or ‘‘modelines’’ edit option. This feature allows any line in the first or last five lines of the file containing the strings `"ex:"` or `"vi:"` (and, apparently, `"ei:"` or `"vx:"`) 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 the editor when replying to a mail message in which a line such as:

```
ex:! rm –rf :
```

appeared in the signature lines. The standard developers believed strongly that an editor should not by default interpret any lines of a file. Vendors are strongly urged to delete this feature from their implementations of *ex* and *vi*.

Asynchronous Events

The intention of the phrase ‘‘complete write’’ is that the entire edit buffer be written to stable storage. The note regarding temporary files is intended for implementations that use temporary files to back edit buffers unnamed by the user.

Historically, SIGQUIT was ignored by *ex*, but was the equivalent of the **Q** command in visual mode; that is, it exited visual mode and entered *ex* mode. IEEE Std 1003.1-200x permits, but does not require, this behavior. Historically, SIGINT was often used by *vi* users to terminate text input mode (<control>-C is often easier to enter than <ESC>). Some implementations of *vi* alerted the terminal on this event, and some did not. IEEE Std 1003.1-200x requires that SIGINT behave identically to <ESC>, and that the terminal not be alerted.

Historically, suspending the *ex* editor during text input mode was similar to SIGINT, as completed lines were retained, but any partial line discarded, and the editor returned to command mode. IEEE Std 1003.1-200x is silent on this issue; implementations are encouraged to follow historical practice, where possible.

Historically, the *vi* editor did not treat SIGTSTP as an asynchronous event, and it was therefore impossible to suspend the editor in visual text input mode. There are two major reasons for this. The first is that SIGTSTP is a broadcast signal on UNIX systems, and the chain of events where the shell *exec*s an application that then *exec*s *vi* usually caused confusion for the terminal state if SIGTSTP was delivered to the process group in the default manner. The second was that most implementations of the UNIX *curses* package are not reentrant, and the receipt of SIGTSTP at the wrong time will cause them to crash. IEEE Std 1003.1-200x is silent on this issue; implementations are encouraged to treat suspension as an asynchronous event if possible.

Historically, modifications to the edit buffer made before SIGINT interrupted an operation were retained; that is, anywhere from zero to all of the lines to be modified might have been modified by the time the SIGINT arrived. These changes were not discarded by the arrival of SIGINT. IEEE Std 1003.1-200x permits this behavior, noting that the *undo* command is required to be able to undo these partially completed commands.

The action taken for signals other than SIGINT, SIGCONT, SIGHUP, and SIGTERM is unspecified because some implementations attempt to save the edit buffer in a useful state when other signals are received.

Standard Error

For *ex*/*vi*, diagnostic messages are those messages reported as a result of a failed attempt to invoke *ex* or *vi*, such as invalid options or insufficient resources, or an abnormal termination condition. Diagnostic messages should not be confused with the error messages generated by inappropriate or illegal user commands.

Initialization in ex and vi

If an *ex* command (other than **cd**, **chdir**, or **source**) has a filename argument, one or both of the alternate and current pathnames will be set. Informally, they are set as follows:

1. If the *ex* command is one that replaces the contents of the edit buffer, and it succeeds, the current pathname will be set to the filename argument (the first filename argument in the case of the **next** command) and the alternate pathname will be set to the previous current pathname, if there was one.
2. In the case of the file read/write forms of the **read** and **write** commands, if there is no current pathname, the current pathname will be set to the filename argument.
3. Otherwise, the alternate pathname will be set to the filename argument.

For example, **:edit foo** and **:recover foo**, when successful, set the current pathname, and, if there was a previous current pathname, the alternate pathname. The commands **:write**, **!command**, and **:edit** set neither the current or alternate pathnames. If the **:edit foo** command were to fail for some reason, the alternate pathname would be set. The **read** and **write** commands set the alternate pathname to their *file* argument, unless the current pathname is not set, in which case they set the current pathname to their *file* arguments. The alternate pathname was not historically set by the **:source** command. IEEE Std 1003.1-200x requires conformance to historical practice. Implementations adding commands that take filenames as arguments are encouraged to set the alternate pathname as described here.

Historically, *ex* and *vi* read the **.exrc** file in the *$HOME* directory twice, if the editor was executed in the *$HOME* directory. IEEE Std 1003.1-200x prohibits this behavior.

Historically, the 4 BSD *ex* and *vi* read the *$HOME* and local **.exrc** files if they were owned by the real ID of the user, or the **sourceany** option was set, regardless of other considerations. This was a security problem because it is possible to put normal UNIX system commands inside a **.exrc**

file. IEEE Std 1003.1-200x does not specify the **sourceany** option, and historical implementations are encouraged to delete it.

The **.exrc** files must be owned by the real ID of the user, and not writeable by anyone other than the owner. The appropriate privileges exception is intended to permit users to acquire special privileges, but continue to use the **.exrc** files in their home directories.

System V Release 3.2 and later *vi* implementations added the option **[no]exrc**. The behavior is that local **.exrc** files are read-only if the **exrc** option is set. The default for the **exrc** option was off, so by default, local **.exrc** files were not read. The problem this was intended to solve was that System V permitted users to give away files, so there is no possible ownership or writeability test to ensure that the file is safe. This is still a security problem on systems where users can give away files, but there is nothing additional that IEEE Std 1003.1-200x can do. The implementation-defined exception is intended to permit groups to have local **.exrc** files that are shared by users, by creating pseudo-users to own the shared files.

IEEE Std 1003.1-200x does not mention system-wide *ex* and *vi* start-up files. While they exist in several implementations of *ex* and *vi*, they are not present in any implementations considered historical practice by IEEE Std 1003.1-200x. Implementations that have such files should use them only if they are owned by the real user ID or an appropriate user (for example, root on UNIX systems) and if they are not writeable by any user other than their owner. System-wide start-up files should be read before the *EXINIT* variable, **$HOME/.exrc** or local **.exrc** files are evaluated.

Historically, any *ex* command could be entered in the *EXINIT* variable or the **.exrc** file, although ones requiring that the edit buffer already contain lines of text generally caused historical implementations of the editor to drop core. IEEE Std 1003.1-200x requires that any *ex* command be permitted in the *EXINIT* variable and **.exrc** files, for simplicity of specification and consistency, although many of them will obviously fail under many circumstances.

The initialization of the contents of the edit buffer uses the phrase ‘‘the effect shall be’’ with regard to various *ex* commands. The intent of this phrase is that edit buffer contents loaded during the initialization phase not be lost; that is, loading the edit buffer should fail if the **.exrc** file read in the contents of a file and did not subsequently write the edit buffer. An additional intent of this phrase is to specify that the initial current line and column is set as specified for the individual *ex* commands.

Historically, the **–t** option behaved as if the tag search were a **+***command*; that is, it was executed from the last line of the file specified by the tag. This resulted in the search failing if the pattern was a forward search pattern and the **wrapscan** edit option was not set. IEEE Std 1003.1-200x does not permit this behavior, requiring that the search for the tag pattern be performed on the entire file, and, if not found, that the current line be set to a more reasonable location in the file.

Historically, the empty edit buffer presented for editing when a file was not specified by the user was unnamed. This is permitted by IEEE Std 1003.1-200x; however, implementations are encouraged to provide users a temporary filename for this buffer because it permits them the use of *ex* commands that use the current pathname during temporary edit sessions.

Historically, the file specified using the **–t** option was not part of the current argument list. This practice is permitted by IEEE Std 1003.1-200x; however, implementations are encouraged to include its name in the current argument list for consistency.

Historically, the **–c** command was generally not executed until a file that already exists was edited. IEEE Std 1003.1-200x requires conformance to this historical practice. Commands that could cause the **–c** command to be executed include the *ex* commands **edit**, **next**, **recover**, **rewind**, and **tag**, and the *vi* commands <control>-^ and <control>-]. Historically, reading a file into an edit buffer did not cause the **–c** command to be executed (even though it might set the

current pathname) with the exception that it did cause the –**c** command to be executed if: the editor was in *ex* mode, the edit buffer had no current pathname, the edit buffer was empty, and no read commands had yet been attempted. For consistency and simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior.

Historically, the –**r** option was the same as a normal edit session if there was no recovery information available for the file. This allowed users to enter:

```
vi –r *.c
```

and recover whatever files were recoverable. In some implementations, recovery was attempted only on the first file named, and the file was not entered into the argument list; in others, recovery was attempted for each file named. In addition, some historical implementations ignored –**r** if –**t** was specified or did not support command line *file* arguments with the –**t** option. For consistency and simplicity of specification, IEEE Std 1003.1-200x disallows these special cases, and requires that recovery be attempted the first time each file is edited.

Historically, *vi* initialized the ‘ and ’ marks, but *ex* did not. This meant that if the first command in *ex* mode was **visual** or if an *ex* command was executed first (for example, *vi* +10 *file*), *vi* was entered without the marks being initialized. Because the standard developers believed the marks to be generally useful, and for consistency and simplicity of specification, IEEE Std 1003.1-200x requires that they always be initialized if in open or visual mode, or if in *ex* mode and the edit buffer is not empty. Not initializing it in *ex* mode if the edit buffer is empty is historical practice; however, it has always been possible to set (and use) marks in empty edit buffers in open and visual mode edit sessions.

Addressing

Historically, *ex* and *vi* accepted the additional addressing forms `'\/'` and `'\?'`. They were equivalent to `"//"` and `"??"`, respectively. They are not required by IEEE Std 1003.1-200x, mostly because nobody can remember whether they ever did anything different historically.

Historically, *ex* and *vi* permitted an address of zero for several commands, and permitted the **%** address in empty files for others. For consistency, IEEE Std 1003.1-200x requires support for the former in the few commands where it makes sense, and disallows it otherwise. In addition, because IEEE Std 1003.1-200x requires that **%** be logically equivalent to `"1,$"`, it is also supported where it makes sense and disallowed otherwise.

Historically, the **%** address could not be followed by further addresses. For consistency and simplicity of specification, IEEE Std 1003.1-200x requires that additional addresses be supported.

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.
`3 ---- 2`	Line one (note intermediate negative address).
`1 2 3`	Line six.

Any number of addresses can be provided to commands taking addresses; for example, `"1,2,3,4,5p"` prints lines 4 and 5, because two is the greatest valid number of addresses accepted by the **print** command. This, in combination with the semicolon delimiter, permits users to create commands based on ordered patterns in the file. For example, the command **3;/foo/;+2print** will display the first line after line 3 that contains the pattern *foo*, plus the next two lines. Note that the address **3;** must be evaluated before being discarded because the search

15549 origin for the **/foo/** command depends on this.

15550 Historically, values could be added to addresses by including them after one or more <blank>s;
15551 for example, **3 – 5p** wrote the seventh line of the file, and **/foo/ 5** was the same as **/foo/+5**.
15552 However, only absolute values could be added; for example, **5 /foo/** was an error.
15553 IEEE Std 1003.1-200x requires conformance to historical practice. Address offsets are separately
15554 specified from addresses because they could historically be provided to visual mode search
15555 commands.

15556 Historically, any missing addresses defaulted to the current line. This was true for leading and
15557 trailing comma-delimited addresses, and for trailing semicolon-delimited addresses. For
15558 consistency, IEEE Std 1003.1-200x requires it for leading semicolon addresses as well.

15559 Historically, *ex* and *vi* accepted the `'^'` character as both an address and as a flag offset for
15560 commands. In both cases it was identical to the `'-'` character. IEEE Std 1003.1-200x does not
15561 require or prohibit this behavior.

15562 Historically, the enhancements to basic regular expressions could be used in addressing; for
15563 example, `'~'`, `'\<'`, and `'\>'`. IEEE Std 1003.1-200x requires conformance to historical
15564 practice; that is, that regular expression usage be consistent, and that regular expression
15565 enhancements be supported wherever regular expressions are used.

15566 **Command Line Parsing in ex**

15567 Historical *ex* command parsing was even more complex than that described here.
15568 IEEE Std 1003.1-200x requires the subset of the command parsing that the standard developers
15569 believed was documented and that users could reasonably be expected to use in a portable
15570 fashion, and that was historically consistent between implementations. (The discarded
15571 functionality is obscure, at best.) Historical implementations will require changes in order to
15572 comply with IEEE Std 1003.1-200x; however, users are not expected to notice any of these
15573 changes. Most of the complexity in *ex* parsing is to handle three special termination cases:

15574 1. The **!**, **global**, **v**, and the filter versions of the **read** and **write** commands are delimited by
15575 <newline>s (they can contain vertical-line characters that are usually shell pipes).

15576 2. The **ex**, **edit**, **next**, and **visual** in open and visual mode commands all take *ex* commands,
15577 optionally containing vertical-line characters, as their first arguments.

15578 3. The **s** command takes a regular expression as its first argument, and uses the delimiting
15579 characters to delimit the command.

15580 Historically, vertical-line characters in the **+***command* argument of the **ex**, **edit**, **next**, **vi**, and
15581 **visual** commands, and in the *pattern* and *replacement* parts of the **s** command, did not delimit the
15582 command, and in the filter cases for **read** and **write**, and the **!**, **global**, and **v** commands, they did
15583 not delimit the command at all. For example, the following commands are all valid:

```
15584 :edit +25 | s/abc/ABC/ file.c
15585 :s/ | /PIPE/
15586 :read !spell % | columnate
15587 :global/pattern/p | l
15588 :s/a/b/ | s/c/d | set
```

15589 Historically, empty or <blank> filled lines in **.exrc** files and **source**d files (as well as *EXINIT*
15590 variables and *ex* command scripts) were treated as default commands; that is, **print** commands.
15591 IEEE Std 1003.1-200x specifically requires that they be ignored when encountered in **.exrc** and
15592 **source**d files to eliminate a common source of new user error.

Historically, *ex* commands with multiple adjacent (or <blank>-separated) vertical lines were handled oddly when executed from *ex* mode. For example, the command **|||** <carriage-return>, when the cursor was on line 1, displayed lines 2, 3, and 5 of the file. In addition, the command **|** would only display the line after the next line, instead of the next two lines. The former worked more logically when executed from *vi* mode, and displayed lines 2, 3, and 4. IEEE Std 1003.1-200x requires the *vi* behavior; that is, a single default command and line number increment for each command separator, and trailing <newline>s after vertical-line separators are discarded.

Historically, *ex* permitted a single extra colon as a leading command character; for example, **:g/pattern/:p** was a valid command. IEEE Std 1003.1-200x generalizes this to require that any number of leading colon characters be stripped.

Historically, any prefix of the **delete** command could be followed without intervening <blank>s by a flag character because in the command **d p**, *p* is interpreted as the buffer *p*. IEEE Std 1003.1-200x requires conformance to historical practice.

Historically, the **k** command could be followed by the mark name without intervening <blank>s. IEEE Std 1003.1-200x requires conformance to historical practice.

Historically, the **s** command could be immediately followed by flag and option characters; for example, **s/e/E/|s|sgc3p** was a valid command. However, flag characters could not stand alone; for example, the commands **sp** and **s l** would fail, while the command **sgp** and **s gl** would succeed. (Obviously, the '#' flag character was used as a delimiter character if it followed the command.) Another issue was that option characters had to precede flag characters even when the command was fully specified; for example, the command **s/e/E/pg** would fail, while the command **s/e/E/gp** would succeed. IEEE Std 1003.1-200x requires conformance to historical practice.

Historically, the first command name that had a prefix matching the input from the user was the executed command; for example, **ve**, **ver**, and **vers** all executed the **version** command. Commands were in a specific order, however, so that **a** matched **append**, not **abbreviate**. IEEE Std 1003.1-200x requires conformance to historical practice. The restriction on command search order for implementations with extensions is to avoid the addition of commands such that the historical prefixes would fail to work portably.

Historical implementations of *ex* and *vi* did not correctly handle multiple *ex* commands, separated by vertical-line characters, that entered or exited visual mode or the editor. Because implementations of *vi* exist that do not exhibit this failure mode, IEEE Std 1003.1-200x does not permit it.

The requirement that alphabetic command names consist of all following alphabetic characters up to the next non-alphabetic character means that alphabetic command names must be separated from their arguments by one or more non-alphabetic characters, normally a <blank> or '!' character, except as specified for the exceptions, the **delete**, **k**, and **s** commands.

Historically, the repeated execution of the *ex* default **print** commands (<control>-D, *eof*, <newline>, <carriage-return>) erased any prompting character and displayed the next lines without scrolling the terminal; that is, immediately below any previously displayed lines. This provided a cleaner presentation of the lines in the file for the user. IEEE Std 1003.1-200x does not require this behavior because it may be impossible in some situations; however, implementations are strongly encouraged to provide this semantic if possible.

Historically, it was possible to change files in the middle of a command, and have the rest of the command executed in the new file; for example:

```
:edit +25 file.c | s/abc/ABC/ | 1
```

was a valid command, and the substitution was attempted in the newly edited file. IEEE Std 1003.1-200x requires conformance to historical practice. The following commands are examples that exercise the *ex* parser:

```
echo 'foo | bar' > file1; echo 'foo/bar' > file2;
vi
:edit +1 | s/|/PIPE/ | w file1 | e file2 | 1 | s/\//SLASH/ | wq
```

Historically, there was no protection in editor implementations to avoid *ex* **global**, **v**, **@**, or ***** commands changing edit buffers during execution of their associated commands. Because this would almost invariably result in catastrophic failure of the editor, and implementations exist that do exhibit these problems, IEEE Std 1003.1-200x requires that changing the edit buffer during a **global** or **v** command, or during a **@** or ***** command for which there will be more than a single execution, be an error. Implementations supporting multiple edit buffers simultaneously are strongly encouraged to apply the same semantics to switching between buffers as well.

The *ex* command quoting required by IEEE Std 1003.1-200x is a superset of the quoting in historical implementations of the editor. For example, it was not historically possible to escape a <blank> in a filename; for example, **:edit foo\\\ bar** would report that too many filenames had been entered for the edit command, and there was no method of escaping a <blank> in the first argument of an **edit**, **ex**, **next**, or **visual** command at all. IEEE Std 1003.1-200x extends historical practice, requiring that quoting behavior be made consistent across all *ex* commands, except for the **map**, **unmap**, **abbreviate**, and **unabbreviate** commands, which historically used <control>-V instead of backslashes for quoting. For those four commands, IEEE Std 1003.1-200x requires conformance to historical practice.

Backslash quoting in *ex* is non-intuitive. Backslash escapes are ignored unless they escape a special character; for example, when performing *file* argument expansion, the string `"\\%"` is equivalent to `'\%'`, not `"\<current path name>"`. This can be confusing for users because backslash is usually one of the characters that causes shell expansion to be performed, and therefore shell quoting rules must be taken into consideration. Generally, quoting characters are only considered if they escape a special character, and a quoting character must be provided for each layer of parsing for which the character is special. As another example, only a single backslash is necessary for the `'\l'` sequence in substitute replacement patterns, because the character `'l'` is not special to any parsing layer above it.

<control>-V quoting in *ex* is slightly different from backslash quoting. In the four commands where <control>-V quoting applies (**abbreviate**, **unabbreviate**, **map**, and **unmap**), any character may be escaped by a <control>-V whether it would have a special meaning or not. IEEE Std 1003.1-200x requires conformance to historical practice.

Historical implementations of the editor did not require delimiters within character classes to be escaped; for example, the command **:s/[/]//** on the string `"xxx/yyy"` would delete the `'//'` from the string. IEEE Std 1003.1-200x disallows this historical practice for consistency and because it places a large burden on implementations by requiring that knowledge of regular expressions be built into the editor parser.

Historically, quoting <newline>s in *ex* commands was handled inconsistently. In most cases, the <newline> always terminated the command, regardless of any preceding escape character, because backslash characters did not escape <newline>s for most *ex* commands. However, some *ex* commands (for example, **s**, **map**, and **abbreviation**) permitted <newline>s to be escaped (although in the case of **map** and **abbreviation**, <control>-V characters escaped them instead of backslashes). This was true in not only the command line, but also **.exrc** and **source**d files. For example, the command:

```
map = foo<control-V><newline>bar
```

would succeed, although it was sometimes difficult to get the <control>-V and the inserted <newline> passed to the *ex* parser. For consistency and simplicity of specification, IEEE Std 1003.1-200x requires that it be possible to escape <newline>s in *ex* commands at all times, using backslashes for most *ex* commands, and using <control>-V characters for the **map** and **abbreviation** commands. For example, the command **print**<newline>**list** is required to be parsed as the single command **print**<newline>**list**. While this differs from historical practice, IEEE Std 1003.1-200x developers believed it unlikely that any script or user depended on the historical behavior.

Historically, an error in a command specified using the **–c** option did not cause the rest of the **–c** commands to be discarded. IEEE Std 1003.1-200x disallows this for consistency with mapped keys, the **@**, **global**, **source**, and **v** commands, the *EXINIT* environment variable, and the **.exrc** files.

Input Editing in ex

One of the common uses of the historical *ex* editor is over slow network connections. Editors that run in canonical mode can require far less traffic to and from, and far less processing on, the host machine, as well as more easily supporting block-mode terminals. For these reasons, IEEE Std 1003.1-200x requires that *ex* be implemented using canonical mode input processing, as was done historically.

IEEE Std 1003.1-200x does not require the historical 4 BSD input editing characters ‘‘word erase’’ or ‘‘literal next’’. For this reason, it is unspecified how they are handled by *ex*, although they must have the required effect. Implementations that resolve them after the line has been ended using a <newline> or <control>-M character, and implementations that rely on the underlying system terminal support for this processing, are both conforming. Implementations are strongly urged to use the underlying system functionality, if at all possible, for compatibility with other system text input interfaces.

Historically, when the *eof* character was used to decrement the **autoindent** level, the cursor moved to display the new end of the **autoindent** characters, but did not move the cursor to a new line, nor did it erase the <control>-D character from the line. IEEE Std 1003.1-200x does not specify that the cursor remain on the same line or that the rest of the line is erased; however, implementations are strongly encouraged to provide the best possible user interface; that is, the cursor should remain on the same line, and any <control>-D character on the line should be erased.

IEEE Std 1003.1-200x does not require the historical 4 BSD input editing character ‘‘reprint’’, traditionally <control>-R, which redisplayed the current input from the user. For this reason, and because the functionality cannot be implemented after the line has been terminated by the user, IEEE Std 1003.1-200x makes no requirements about this functionality. Implementations are strongly urged to make this historical functionality available, if possible.

Historically, <control>-Q did not perform a literal next function in *ex*, as it did in *vi*. IEEE Std 1003.1-200x requires conformance to historical practice to avoid breaking historical *ex* scripts and **.exrc** files.

15728 **eof**

15729 Whether the *eof* character immediately modifies the **autoindent** characters in the prompt is left
15730 unspecified so that implementations can conform in the presence of systems that do not support
15731 this functionality. Implementations are encouraged to modify the line and redisplay it
15732 immediately, if possible.

15733 The specification of the handling of the *eof* character differs from historical practice only in that
15734 *eof* characters are not discarded if they follow normal characters in the text input. Historically,
15735 they were always discarded.

15736 **Command Descriptions in ex**

15737 Historically, several commands (for example, **global**, **v**, **visual**, **s**, **write**, **wq**, **yank**, **!**, **<**, **>**, **&**, and
15738 **~**) were executable in empty files (that is, the default address(es) were 0), or permitted explicit
15739 addresses of 0 (for example, 0 was a valid address, or 0,0 was a valid range). Addresses of 0, or
15740 command execution in an empty file, make sense only for commands that add new text to the
15741 edit buffer or write commands (because users may wish to write empty files).
15742 IEEE Std 1003.1-200x requires this behavior for such commands and disallows it otherwise, for
15743 consistency and simplicity of specification.

15744 A count to an *ex* command has been historically corrected to be no greater than the last line in a
15745 file; for example, in a five-line file, the command **1,6print** would fail, but the command **1print300**
15746 would succeed. IEEE Std 1003.1-200x requires conformance to historical practice.

15747 Historically, the use of flags in *ex* commands could be obscure. General historical practice was as
15748 described by IEEE Std 1003.1-200x, but there were some special cases. For example, the **list**,
15749 **number**, and **print** commands ignored trailing address offsets; for example, **3p +++#** would
15750 display line 3, and 3 would be the current line after the execution of the command. The **open** and
15751 **visual** commands ignored both the trailing offsets and the trailing flags. Also, flags specified to
15752 the **open** and **visual** commands interacted badly with the **list** edit option, and setting and then
15753 unsetting it during the open/visual session would cause *vi* to stop displaying lines in the
15754 specified format. For consistency and simplicity of specification, IEEE Std 1003.1-200x does not
15755 permit any of these exceptions to the general rule.

15756 IEEE Std 1003.1-200x uses the word *copy* in several places when discussing buffers. This is not
15757 intended to imply implementation.

15758 Historically, *ex* users could not specify numeric buffers because of the ambiguity this would
15759 cause; for example, in the command **3 delete 2**, it is unclear whether 2 is a buffer name or a
15760 *count*. IEEE Std 1003.1-200x requires conformance to historical practice by default, but does not
15761 preclude extensions.

15762 Historically, the contents of the unnamed buffer were frequently discarded after commands that
15763 did not explicitly affect it; for example, when using the **edit** command to switch files. For
15764 consistency and simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior.

15765 The *ex* utility did not historically have access to the numeric buffers, and, furthermore, deleting
15766 lines in *ex* did not modify their contents. For example, if, after doing a delete in *vi*, the user
15767 switched to *ex*, did another delete, and then switched back to *vi*, the contents of the numeric
15768 buffers would not have changed. IEEE Std 1003.1-200x requires conformance to historical
15769 practice. Numeric buffers are described in the *ex* utility in order to confine the description of
15770 buffers to a single location in IEEE Std 1003.1-200x.

15771 The metacharacters that trigger shell expansion in *file* arguments match historical practice, as
15772 does the method for doing shell expansion. Implementations wishing to provide users with the
15773 flexibility to alter the set of metacharacters are encouraged to provide a **shellmeta** string edit

15774 option.

15775 Historically, *ex* commands executed from *vi* refreshed the screen when it did not strictly need to
15776 do so; for example, **:!date > /dev/null** does not require a screen refresh because the output of the
15777 UNIX *date* command requires only a single line of the screen. IEEE Std 1003.1-200x requires that
15778 the screen be refreshed if it has been overwritten, but makes no requirements as to how an
15779 implementation should make that determination. Implementations may prompt and refresh the
15780 screen regardless.

15781 **Abbreviate**

15782 Historical practice was that characters that were entered as part of an abbreviation replacement
15783 were subject to **map** expansions, the **showmatch** edit option, further abbreviation expansions,
15784 and so on; that is, they were logically pushed onto the terminal input queue, and were not a
15785 simple replacement. IEEE Std 1003.1-200x requires conformance to historical practice. Historical
15786 practice was that whenever a non-word character (that had not been escaped by a <control>-V)
15787 was entered after a word character, *vi* would check for abbreviations. The check was based on
15788 the type of the character entered before the word character of the word/non-word pair that
15789 triggered the check. The word character of the word/non-word pair that triggered the check and
15790 all characters entered before the trigger pair that were of that type were included in the check,
15791 with the exception of <blank>s, which always delimited the abbreviation.

15792 This means that, for the abbreviation to work, the *lhs* must end with a word character, there can
15793 be no transitions from word to non-word characters (or *vice versar*) other than between the last
15794 and next-to-last characters in the *lhs*, and there can be no <blank>s in the *lhs*. In addition,
15795 because of the historical quoting rules, it was impossible to enter a literal <control>-V in the *lhs*.
15796 IEEE Std 1003.1-200x requires conformance to historical practice. Historical implementations did
15797 not inform users when abbreviations that could never be used were entered; implementations
15798 are strongly encouraged to do so.

15799 For example, the following abbreviations will work:

15800 ```
:ab (p REPLACE
15801 :ab p   REPLACE
15802 :ab ((p REPLACE
```

15803 The following abbreviations will not work:

15804 ```
:ab (   REPLACE
15805 :ab (pp REPLACE
```

15806 Historical practice is that words on the *vi* colon command line were subject to abbreviation
15807 expansion, including the arguments to the **abbrev** (and more interestingly) the **unabbrev**
15808 command. Because there are implementations that do not do abbreviation expansion for the first
15809 argument to those commands, this is permitted, but not required, by IEEE Std 1003.1-200x.
15810 However, the following sequence:

15811 ```
:ab foo bar
15812 :ab foo baz
```

15813 resulted in the addition of an abbreviation of `"baz"` for the string `"bar"` in historical *ex*/*vi*, and
15814 the sequence:

15815 ```
:ab foo1 bar
15816 :ab foo2 bar
15817 :unabbreviate foo2
```

15818 deleted the abbreviation `"foo1"`, not `"foo2"`. These behaviors are not permitted by
15819 IEEE Std 1003.1-200x because they clearly violate the expectations of the user.

15820 It was historical practice that <control>-V, not backslash, characters be interpreted as escaping
15821 subsequent characters in the **abbreviate** command. IEEE Std 1003.1-200x requires conformance
15822 to historical practice; however, it should be noted that an abbreviation containing a <blank> will
15823 never work.

15824 **Append**

15825 Historically, any text following a vertical-line command separator after an **append**, **change**, or
15826 **insert** command became part of the insert text. For example, in the command:

15827 `:g/pattern/append|stuff1`

15828 a line containing the text `"stuff1"` would be appended to each line matching pattern. It was
15829 also historically valid to enter:

15830 `:append|stuff1`
15831 `stuff2`
15832 `.`

15833 and the text on the *ex* command line would be appended along with the text inserted after it.
15834 There was an historical bug, however, that the user had to enter two terminating lines (the `'.'`
15835 lines) to terminate text input mode in this case. IEEE Std 1003.1-200x requires conformance to
15836 historical practice, but disallows the historical need for multiple terminating lines.

15837 **Change**

15838 See the RATIONALE for the **append** command. Historical practice for cursor positioning after
15839 the change command when no text is input, is as described in IEEE Std 1003.1-200x. However,
15840 one System V implementation is known to have been modified such that the cursor is positioned
15841 on the first address specified, and not on the line before the first address. IEEE Std 1003.1-200x
15842 disallows this modification for consistency.

15843 Historically, the **change** command did not support buffer arguments, although some
15844 implementations allow the specification of an optional buffer. This behavior is neither required
15845 nor disallowed by IEEE Std 1003.1-200x.

15846 **Change Directory**

15847 A common extension in *ex* implementations is to use the elements of a **cdpath** edit option as
15848 prefix directories for *path* arguments to **chdir** that are relative pathnames and that do not have
15849 `'.'` or `".."` as their first component. Elements in the **cdpath** edit option are colon-separated.
15850 The initial value of the **cdpath** edit option is the value of the shell *CDPATH* environment
15851 variable. This feature was not included in IEEE Std 1003.1-200x because it does not exist in any
15852 of the implementations considered historical practice.

15853 **Copy**

15854 Historical implementations of *ex* permitted copies to lines inside of the specified range; for
15855 example, **:2,5copy3** was a valid command. IEEE Std 1003.1-200x requires conformance to
15856 historical practice.

15857 **Delete**

15858 IEEE Std 1003.1-200x requires support for the historical parsing of a **delete** command followed
15859 by flags, without any intervening <blank>s. For example:

15860 **1dp** Deletes the first line and prints the line that was second.

15861 **1delep** As for **1dp**.

15862 **1d** Deletes the first line, saving it in buffer *p*.

15863 **1d p1l** (Pee-one-ell.) Deletes the first line, saving it in buffer *p*, and listing the line that was
15864 second.

15865 **Edit**

15866 Historically, any *ex* command could be entered as a **+***command* argument to the **edit** command,
15867 although some (for example, **insert** and **append**) were known to confuse historical
15868 implementations. For consistency and simplicity of specification, IEEE Std 1003.1-200x requires
15869 that any command be supported as an argument to the **edit** command.

15870 Historically, the command argument was executed with the current line set to the last line of the
15871 file, regardless of whether the **edit** command was executed from visual mode or not.
15872 IEEE Std 1003.1-200x requires conformance to historical practice.

15873 Historically, the **+***command* specified to the **edit** and **next** commands was delimited by the first
15874 <blank>, and there was no way to quote them. For consistency, IEEE Std 1003.1-200x requires
15875 that the usual *ex* backslash quoting be provided.

15876 Historically, specifying the **+***command* argument to the edit command required a filename to be
15877 specified as well; for example, **:edit +100** would always fail. For consistency and simplicity of
15878 specification, IEEE Std 1003.1-200x does not permit this usage to fail for that reason.

15879 Historically, only the cursor position of the last file edited was remembered by the editor.
15880 IEEE Std 1003.1-200x requires that this be supported; however, implementations are permitted
15881 to remember and restore the cursor position for any file previously edited.

15882 **File**

15883 Historical versions of the *ex* editor **file** command displayed a current line and number of lines in
15884 the edit buffer of 0 when the file was empty, while the *vi* <control>-G command displayed a
15885 current line and number of lines in the edit buffer of 1 in the same situation.
15886 IEEE Std 1003.1-200x does not permit this discrepancy, instead requiring that a message be
15887 displayed indicating that the file is empty.

15888 **Global**

15889 The two-pass operation of the **global** and **v** commands is not intended to imply implementation,
15890 only the required result of the operation.

15891 The current line and column are set as specified for the individual *ex* commands. This
15892 requirement is cumulative; that is, the current line and column must track across all the
15893 commands executed by the **global** or **v** commands.

15894 **Insert**

15895 See the RATIONALE for the **append** command.

15896 Historically, **insert** could not be used with an address of zero; that is, not when the edit buffer
15897 was empty. IEEE Std 1003.1-200x requires that this command behave consistently with the
15898 **append** command.

15899 **Join**

15900 The action of the **join** command in relation to the special characters is only defined for the
15901 POSIX locale because the correct amount of white space after a period varies; in Japanese none is
15902 required, in French only a single space, and so on.

15903 **List**

15904 The historical output of the **list** command was potentially ambiguous. The standard developers
15905 believed correcting this to be more important than adhering to historical practice, and
15906 IEEE Std 1003.1-200x requires unambiguous output.

15907 **Map**

15908 Historically, command mode maps only applied to command names; for example, if the
15909 character `'x'` was mapped to `'y'`, the command **fx** searched for the `'x'` character, not the `'y'`
15910 character. IEEE Std 1003.1-200x requires this behavior. Historically, entering <control>-V as the
15911 first character of a *vi* command was an error. Several implementations have extended the
15912 semantics of *vi* such that <control>-V means that the subsequent command character is not
15913 mapped. This is permitted, but not required, by IEEE Std 1003.1-200x. Regardless, using
15914 <control>-V to escape the second or later character in a sequence of characters that might match
15915 a **map** command, or any character in text input mode, is historical practice, and stops the entered
15916 keys from matching a map. IEEE Std 1003.1-200x requires conformance to historical practice.

15917 Historical implementations permitted digits to be used as a **map** command *lhs*, but then ignored
15918 the map. IEEE Std 1003.1-200x requires that the mapped digits not be ignored.

15919 The historical implementation of the **map** command did not permit **map** commands that were
15920 more than a single character in length if the first character was printable. This behavior is
15921 permitted, but not required, by IEEE Std 1003.1-200x.

15922 Historically, mapped characters were remapped unless the **remap** edit option was not set, or the
15923 prefix of the mapped characters matched the mapping characters; for example, in the **map**:

15924 `:map ab abcd`

15925 the characters `"ab"` were used as is and were not remapped, but the characters `"cd"` were
15926 mapped if appropriate. This can cause infinite loops in the *vi* mapping mechanisms.
15927 IEEE Std 1003.1-200x requires conformance to historical practice, and that such loops be
15928 interruptible.

15929 Text input maps had the same problems with expanding the *lhs* for the *ex* **map!** and **unmap!**
15930 command as did the *ex* **abbreviate** and **unabbreviate** commands. See the RATIONALE for the *ex*
15931 **abbreviate** command. IEEE Std 1003.1-200x requires similar modification of some historical
15932 practice for the **map** and **unmap** commands, as described for the **abbreviate** and **unabbreviate**
15933 commands.

15934 Historically, **map**s that were subsets of other **map**s behaved differently depending on the order
15935 in which they were defined. For example:

```
:map! ab      short
:map! abc     long
```

would always translate the characters "ab" to "short", regardless of how fast the characters "abc" were entered. If the entry order was reversed:

```
:map! abc     long
:map! ab      short
```

the characters "ab" would cause the editor to pause, waiting for the completing 'c' character, and the characters might never be mapped to "short". For consistency and simplicity of specification, IEEE Std 1003.1-200x requires that the shortest match be used at all times.

The length of time the editor spends waiting for the characters to complete the *lhs* is unspecified because the timing capabilities of systems are often inexact and variable, and it may depend on other factors such as the speed of the connection. The time should be long enough for the user to be able to complete the sequence, but not long enough for the user to have to wait. Some implementations of *vi* have added a **keytime** option, which permits users to set the number of 0,1 seconds the editor waits for the completing characters. Because mapped terminal function and cursor keys tend to start with an <ESC> character, and <ESC> is the key ending *vi* text input mode, **map**s starting with <ESC> characters are generally exempted from this timeout period, or, at least timed out differently.

Mark

Historically, users were able to set the ''previous context'' marks explicitly. In addition, the *ex* commands **''** and **`** and the *vi* commands **''**, **``**, **`'**, and **`** all referred to the same mark. In addition, the previous context marks were not set if the command, with which the address setting the mark was associated, failed. IEEE Std 1003.1-200x requires conformance to historical practice. Historically, if marked lines were deleted, the mark was also deleted, but would reappear if the change was undone. IEEE Std 1003.1-200x requires conformance to historical practice.

The description of the special events that set the **`** and **'** marks matches historical practice. For example, historically the command **/a/,/b/** did not set the **`** and **'** marks, but the command **/a/,/b/delete** did.

Next

Historically, any *ex* command could be entered as a **+***command* argument to the **next** command, although some (for example, **insert** and **append**) were known to confuse historical implementations. IEEE Std 1003.1-200x requires that any command be permitted and that it behave as specified. The **next** command can accept more than one file, so usage such as:

```
next `ls [abc] `
```

is valid; it need not be valid for the **edit** or **read** commands, for example, because they expect only one filename.

Historically, the **next** command behaved differently from the **:rewind** command in that it ignored the force flag if the **autowrite** flag was set. For consistency, IEEE Std 1003.1-200x does not permit this behavior.

Historically, the **next** command positioned the cursor as if the file had never been edited before, regardless. IEEE Std 1003.1-200x does not permit this behavior, for consistency with the **edit** command.

Implementations wanting to provide a counterpart to the **next** command that edited the previous file have used the command **prev[ious],** which takes no *file* argument.

IEEE Std 1003.1-200x does not require this command.

Open

Historically, the **open** command would fail if the **open** edit option was not set. IEEE Std 1003.1-200x does not mention the **open** edit option and does not require this behavior. Some historical implementations do not permit entering open mode from open or visual mode, only from *ex* mode. For consistency, IEEE Std 1003.1-200x does not permit this behavior.

Historically, entering open mode from the command line (that is, *vi* **+open**) resulted in anomalous behaviors; for example, the *ex* file and *set* commands, and the *vi* command <control>-G did not work. For consistency, IEEE Std 1003.1-200x does not permit this behavior.

Historically, the **open** command only permitted `'//'` characters to be used as the search pattern delimiter. For consistency, IEEE Std 1003.1-200x requires that the search delimiters used by the **s**, **global**, and **v** commands be accepted as well.

Preserve

The **preserve** command does not historically cause the file to be considered unmodified for the purposes of future commands that may exit the editor. IEEE Std 1003.1-200x requires conformance to historical practice.

Historical documentation stated that mail was not sent to the user when preserve was executed; however, historical implementations did send mail in this case. IEEE Std 1003.1-200x requires conformance to the historical implementations.

Print

The writing of NUL by the **print** command is not specified as a special case because the standard developers did not want to require *ex* to support NUL characters. Historically, characters were displayed using the ARPA standard mappings, which are as follows:

1. Printable characters are left alone.
2. Control characters less than \177 are represented as `'^'` followed by the character offset from the `'@'` character in the ASCII map; for example, \007 is represented as `'^G'`.
3. \177 is represented as `'^'` followed by `'?'`.

The display of characters having their eighth bit set was less standard. Existing implementations use hex (0x00), octal (\000), and a meta-bit display. (The latter displayed bytes that had their eighth bit set as the two characters `"M-"` followed by the seven-bit display as described above.) The latter probably has the best claim to historical practice because it was used for the **–v** option of 4 BSD and 4 BSD-derived versions of the *cat* utility since 1980.

No specific display format is required by IEEE Std 1003.1-200x.

Explicit dependence on the ASCII character set has been avoided where possible, hence the use of the phrase an ‘‘implementation-defined multi-character sequence’’ for the display of non-printable characters in preference to the historical usage of, for instance, `"^I"` for the <tab>. Implementations are encouraged to conform to historical practice in the absence of any strong reason to diverge.

Historically, all *ex* commands beginning with the letter `'p'` could be entered using capitalized versions of the commands; for example, **P[rint]**, **Pre[serve]**, and **Pu[t]** were all valid command names. IEEE Std 1003.1-200x permits, but does not require, this historical practice because capital forms of the commands are used by some implementations for other purposes.

16022 **Put**

16023 Historically, an *ex* **put** command, executed from open or visual mode, was the same as the open
16024 or visual mode **P** command, if the buffer was named and was cut in character mode, and the
16025 same as the **p** command if the buffer was named and cut in line mode. If the unnamed buffer
16026 was the source of the text, the entire line from which the text was taken was usually **put**, and the
16027 buffer was handled as if in line mode, but it was possible to get extremely anomalous behavior.
16028 In addition, using the **Q** command to switch into *ex* mode, and then doing a **put** often resulted in
16029 errors as well, such as appending text that was unrelated to the (supposed) contents of the
16030 buffer. For consistency and simplicity of specification, IEEE Std 1003.1-200x does not permit
16031 these behaviors. All *ex* **put** commands are required to operate in line mode, and the contents of
16032 the buffers are not altered by changing the mode of the editor.

16033 **Read**

16034 Historically, an *ex* **read** command executed from open or visual mode, executed in an empty file,
16035 left an empty line as the first line of the file. For consistency and simplicity of specification,
16036 IEEE Std 1003.1-200x does not permit this behavior. Historically, a **read** in open or visual mode
16037 from a program left the cursor at the last line read in, not the first. For consistency,
16038 IEEE Std 1003.1-200x does not permit this behavior.

16039 Historical implementations of *ex* were unable to undo **read** commands that read from the output
16040 of a program. For consistency, IEEE Std 1003.1-200x does not permit this behavior.

16041 Historically, the *ex* and *vi* message after a successful **read** or **write** command specified
16042 ‘‘characters’’, not ‘‘bytes’’. IEEE Std 1003.1-200x requires that the number of bytes be displayed,
16043 not the number of characters, because it may be difficult in multi-byte implementations to
16044 determine the number of characters read. Implementations are encouraged to clarify the
16045 message displayed to the user.

16046 Historically, reads were not permitted on files other than type regular, except that FIFO files
16047 could be read (probably only because they did not exist when *ex* and *vi* were originally written).
16048 Because the historical *ex* evaluated **read!** and **read !** equivalently, there can be no optional way
16049 to force the read. IEEE Std 1003.1-200x permits, but does not require, this behavior.

16050 **Recover**

16051 Some historical implementations of the editor permitted users to recover the edit buffer contents
16052 from a previous edit session, and then exit without saving those contents (or explicitly
16053 discarding them). The intent of IEEE Std 1003.1-200x in requiring that the edit buffer be treated
16054 as already modified is to prevent this user error.

16055 **Rewind**

16056 Historical implementations supported the **rewind** command when the user was editing the first
16057 file in the list; that is, the file that the **rewind** command would edit. IEEE Std 1003.1-200x
16058 requires conformance to historical practice.

Substitute

Historically, *ex* accepted an **r** option to the **s** command. The effect of the **r** option was to use the last regular expression used in any command as the pattern, the same as the **~** command. The **r** option is not required by IEEE Std 1003.1-200x. Historically, the **c** and **g** options were toggled; for example, the command **:s/abc/def/** was the same as **s/abc/def/ccccgggg**. For simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior.

The tilde command is often used to replace the last search RE. For example, in the sequence:

```
s/red/blue/
/green
~
```

the **~** command is equivalent to:

```
s/green/blue/
```

Historically, *ex* accepted all of the following forms:

```
s/abc/def/
s/abc/def
s/abc/
s/abc
```

IEEE Std 1003.1-200x requires conformance to this historical practice.

The **s** command presumes that the `'^'` character only occupies a single column in the display. Much of the *ex* and *vi* specification presumes that the <space> only occupies a single column in the display. There are no known character sets for which this is not true.

Historically, the final column position for the substitute commands was based on previous column movements; a search for a pattern followed by a substitution would leave the column position unchanged, while a 0 command followed by a substitution would change the column position to the first non-<blank>. For consistency and simplicity of specification, IEEE Std 1003.1-200x requires that the final column position always be set to the first non-<blank>.

Set

Historical implementations redisplayed all of the options for each occurrence of the **all** keyword. IEEE Std 1003.1-200x permits, but does not require, this behavior.

Tag

No requirement is made as to where *ex* and *vi* shall look for the file referenced by the tag entry. Historical practice has been to look for the path found in the **tags** file, based on the current directory. A useful extension found in some implementations is to look based on the directory containing the tags file that held the entry, as well. No requirement is made as to which reference for the tag in the tags file is used. This is deliberate, in order to permit extensions such as multiple entries in a tags file for a tag.

Because users often specify many different tags files, some of which need not be relevant or exist at any particular time, IEEE Std 1003.1-200x requires that error messages about problem tags files be displayed only if the requested tag is not found, and then, only once for each time that the **tag** edit option is changed.

The requirement that the current edit buffer be unmodified is only necessary if the file indicated by the tag entry is not the same as the current file (as defined by the current pathname).

Historically, the file would be reloaded if the filename had changed, as well as if the filename was different from the current pathname. For consistency and simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior, requiring that the name be the only factor in the decision.

Historically, *vi* only searched for tags in the current file from the current cursor to the end of the file, and therefore, if the **wrapscan** option was not set, tags occurring before the current cursor were not found. IEEE Std 1003.1-200x considers this a bug, and implementations are required to search for the first occurrence in the file, regardless.

Undo

The **undo** description deliberately uses the word ''modified''. The **undo** command is not intended to undo commands that replace the contents of the edit buffer, such as **edit**, **next**, **tag**, or **recover**.

Cursor positioning after the **undo** command was inconsistent in the historical *vi*, sometimes attempting to restore the original cursor position (**global**, **undo**, and **v** commands), and sometimes, in the presence of maps, placing the cursor on the last line added or changed instead of the first. IEEE Std 1003.1-200x requires a simplified behavior for consistency and simplicity of specification.

Version

The **version** command cannot be exactly specified since there is no widely-accepted definition of what the version information should contain. Implementations are encouraged to do something reasonably intelligent.

Write

Historically, the *ex* and *vi* message after a successful **read** or **write** command specified ''characters'', not ''bytes''. IEEE Std 1003.1-200x requires that the number of bytes be displayed, not the number of characters because it may be difficult in multi-byte implementations to determine the number of characters written. Implementations are encouraged to clarify the message displayed to the user.

Implementation-defined tests are permitted so that implementations can make additional checks; for example, for locks or file modification times.

Historically, attempting to append to a nonexistent file caused an error. It has been left unspecified in IEEE Std 1003.1-200x to permit implementations to let the **write** succeed, so that the append semantics are similar to those of the historical *csh*.

Historical *vi* permitted empty edit buffers to be written. However, since the way *vi* got around dealing with ''empty'' files was to always have a line in the edit buffer, no matter what, it wrote them as files of a single, empty line. IEEE Std 1003.1-200x does not permit this behavior.

Historically, *ex* restored standard output and standard error to their values as of when *ex* was invoked, before writes to programs were performed. This could disturb the terminal configuration as well as be a security issue for some terminals. IEEE Std 1003.1-200x does not permit this, requiring that the program output be captured and displayed as if by the *ex* **print** command.

16142 **Adjust Window**

16143 Historically, the line count was set to the value of the **scroll** option if the type character was
16144 end-of-file. This feature was broken on most historical implementations long ago, however, and
16145 is not documented anywhere. For this reason, IEEE Std 1003.1-200x is resolutely silent.

16146 Historically, the **z** command was <blank>-sensitive and **z +** and **z –** did different things than **z+**
16147 and **z–** because the type could not be distinguished from a flag. (The commands **z .** and **z =**
16148 were historically invalid.) IEEE Std 1003.1-200x requires conformance to this historical practice.

16149 Historically, the **z** command was further <blank>-sensitive in that the *count* could not be
16150 <blank>-delimited; for example, the commands **z= 5** and **z– 5** were also invalid. Because the
16151 *count* is not ambiguous with respect to either the type character or the flags, this is not permitted
16152 by IEEE Std 1003.1-200x.

16153 **Escape**

16154 Historically, *ex* filter commands only read the standard output of the commands, letting
16155 standard error appear on the terminal as usual. The *vi* utility, however, read both standard
16156 output and standard error. IEEE Std 1003.1-200x requires the latter behavior for both *ex* and *vi*,
16157 for consistency.

16158 **Shift Left and Shift Right**

16159 Historically, it was possible to add shift characters to increase the effect of the command; for
16160 example, **<<<** outdented (or **>>>** indented) the lines 3 levels of indentation instead of the default
16161 1. IEEE Std 1003.1-200x requires conformance to historical practice.

16162 **<control>-D**

16163 Historically, the <control>-D command erased the prompt, providing the user with an unbroken
16164 presentation of lines from the edit buffer. This is not required by IEEE Std 1003.1-200x;
16165 implementations are encouraged to provide it if possible. Historically, the <control>-D
16166 command took, and then ignored, a *count*. IEEE Std 1003.1-200x does not permit this behavior.

16167 **Write Line Number**

16168 Historically, the *ex* **=** command, when executed in *ex* mode in an empty edit buffer, reported 0,
16169 and from open or visual mode, reported 1. For consistency and simplicity of specification,
16170 IEEE Std 1003.1-200x does not permit this behavior.

16171 **Execute**

16172 Historically, *ex* did not correctly handle the inclusion of text input commands (that is, **append**,
16173 **insert**, and **change**) in executed buffers. IEEE Std 1003.1-200x does not permit this exclusion for
16174 consistency.

16175 Historically, the logical contents of the buffer being executed did not change if the buffer itself
16176 were modified by the commands being executed; that is, buffer execution did not support self-
16177 modifying code. IEEE Std 1003.1-200x requires conformance to historical practice.

16178 Historically, the **@** command took a range of lines, and the **@** buffer was executed once per line,
16179 with the current line (`'.'`) set to each specified line. IEEE Std 1003.1-200x requires conformance
16180 to historical practice.

16181 Some historical implementations did not notice if errors occurred during buffer execution. This,
16182 coupled with the ability to specify a range of lines for the *ex* **@** command, makes it trivial to
16183 cause them to drop core. IEEE Std 1003.1-200x requires that implementations stop buffer

execution if any error occurs, if the specified line doesn't exist, or if the contents of the edit buffer itself are replaced (for example, the buffer executes the *ex* **:edit** command).

Regular Expressions in ex

Historical practice is that the characters in the replacement part of the last **s** command—that is, those matched by entering a `'~'` in the regular expression—were not further expanded by the regular expression engine. So, if the characters contained the string `"a.,"` they would match `'a'` followed by `".,"` and not `'a'` followed by any character. IEEE Std 1003.1-200x requires conformance to historical practice.

Edit Options in ex

The following paragraphs describe the historical behavior of some edit options that were not, for whatever reason, included in IEEE Std 1003.1-200x. Implementations are strongly encouraged to only use these names if the functionality described here is fully supported.

extended The **extended** edit option has been used in some implementations of *vi* to provide extended regular expressions instead of basic regular expressions This option was omitted from IEEE Std 1003.1-200x because it is not widespread historical practice.

flash The **flash** edit option historically caused the screen to flash instead of beeping on error. This option was omitted from IEEE Std 1003.1-200x because it is not found in some historical implementations.

hardtabs The **hardtabs** edit option historically defined the number of columns between hardware tab settings. This option was omitted from IEEE Std 1003.1-200x because it was believed to no longer be generally useful.

modeline The **modeline** (sometimes named **modelines**) edit option historically caused *ex* or *vi* to read the five first and last lines of the file for editor commands. This option is a security problem, and vendors are strongly encouraged to delete it from historical implementations.

open The **open** edit option historically disallowed the *ex* **open** and **visual** commands. This edit option was omitted because these commands are required by IEEE Std 1003.1-200x.

optimize The **optimize** edit option historically expedited text throughput by setting the terminal to not do automatic carriage returns when printing more than one logical line of output. This option was omitted from IEEE Std 1003.1-200x because it was intended for terminals without addressable cursors, which are rarely, if ever, still used.

ruler The **ruler** edit option has been used in some implementations of *vi* to present a current row/column ruler for the user. This option was omitted from IEEE Std 1003.1-200x because it is not widespread historical practice.

sourceany The **sourceany** edit option historically caused *ex* or *vi* to source start-up files that were owned by users other than the user running the editor. This option is a security problem, and vendors are strongly encouraged to remove it from their implementations.

timeout The **timeout** edit option historically enabled the (now standard) feature of only waiting for a short period before returning keys that could be part of a macro. This feature was omitted from IEEE Std 1003.1-200x because its behavior is now standard, it is not widely useful, and it was rarely documented.

16228 **verbose** The **verbose** edit option has been used in some implementations of *vi* to cause *vi* to
16229 output error messages for common errors; for example, attempting to move the
16230 cursor past the beginning or end of the line instead of only alerting the screen. (The
16231 historical *vi* only alerted the terminal and presented no message for such errors.
16232 The historical editor option **terse** did not select when to present error messages, it
16233 only made existing error messages more or less verbose.) This option was omitted
16234 from IEEE Std 1003.1-200x because it is not widespread historical practice;
16235 however, implementors are encouraged to use it if they wish to provide error
16236 messages for naive users.

16237 **wraplen** The **wraplen** edit option has been used in some implementations of *vi* to specify an
16238 automatic margin measured from the left margin instead of from the right margin.
16239 This is useful when multiple screen sizes are being used to edit a single file. This
16240 option was omitted from IEEE Std 1003.1-200x because it is not widespread
16241 historical practice; however, implementors are encouraged to use it if they add this
16242 functionality.

16243 **autoindent, ai**

16244 Historically, the command **0a** did not do any autoindentation, regardless of the current
16245 indentation of line 1. IEEE Std 1003.1-200x requires that any indentation present in line 1 be used.

16246 **autoprint, ap**

16247 Historically, the **autoprint** edit option was not completely consistent or based solely on
16248 modifications to the edit buffer. Exceptions were the **read** command (when reading from a file,
16249 but not from a filter), the **append**, **change**, **insert**, **global**, and **v** commands, all of which were not
16250 affected by **autoprint**, and the **tag** command, which was affected by **autoprint**.
16251 IEEE Std 1003.1-200x requires conformance to historical practice.

16252 Historically, the **autoprint** option only applied to the last of multiple commands entered using
16253 vertical-bar delimiters; for example, **delete** <newline> was affected by **autoprint**, but
16254 **delete|version** <newline> was not. IEEE Std 1003.1-200x requires conformance to historical
16255 practice.

16256 **autowrite, aw**

16257 Appending the '!' character to the *ex* **next** command to avoid performing an automatic write
16258 was not supported in historical implementations. IEEE Std 1003.1-200x requires that the
16259 behavior match the other *ex* commands for consistency.

16260 **ignorecase, ic**

16261 Historical implementations of case-insensitive matching (the **ignorecase** edit option) lead to
16262 counterintuitive situations when uppercase characters were used in range expressions.
16263 Historically, the process was as follows:

16264 1. Take a line of text from the edit buffer.

16265 2. Convert uppercase to lowercase in text line.

16266 3. Convert uppercase to lowercase in regular expressions, except in character class
16267 specifications.

16268 4. Match regular expressions against text.

16269 This would mean that, with **ignorecase** in effect, the text:

16270 `The cat sat on the mat`

16271 would be matched by

16272 `/^the/`

16273 but not by:

16274 `/^[A–Z]he/`

16275 For consistency with other commands implementing regular expressions, IEEE Std 1003.1-200x
16276 does not permit this behavior.

16277 **paragraphs, para**

16278 Earlier versions of IEEE Std 1003.1-200x made the default **paragraphs** and **sections** edit options
16279 implementation-defined, arguing they were historically oriented to the UNIX system *troff* text
16280 formatter, and a ‘‘portable user’’ could use the **{**, **}**, **[[**, **]]**, **(**, and **)** commands in open or visual
16281 mode and have the cursor stop in unexpected places. IEEE Std 1003.1-200x specifies their values
16282 in the POSIX locale because the unusual grouping (they only work when grouped into two
16283 characters at a time) means that they cannot be used for general purpose movement, regardless.

16284 **readonly**

16285 Implementations are encouraged to provide the best possible information to the user as to the
16286 read-only status of the file, with the exception that they should not consider the current special
16287 privileges of the process. This provides users a safety net because they must force the overwrite
16288 of read-only files, even when running with additional privileges.

16289 The **readonly** edit option specification largely conforms to historical practice. The only
16290 difference is that historical implementations did not notice that the user had set the **readonly**
16291 edit option in cases where the file was already marked read-only for some reason, and would
16292 therefore reinitialize the **readonly** edit option the next time the contents of the edit buffer were
16293 replaced. This behavior is disallowed by IEEE Std 1003.1-200x.

16294 **report**

16295 The requirement that lines copied to a buffer interact differently than deleted lines is historical
16296 practice. For example, if the **report** edit option is set to 3, deleting 3 lines will cause a report to be
16297 written, but 4 lines must be copied before a report is written.

16298 The requirement that the *ex* **global**, **v**, **open**, **undo**, and **visual** commands present reports based
16299 on the total number of lines added or deleted during the command execution, and that
16300 commands executed by the **global** and **v** commands not present reports, is historical practice.
16301 IEEE Std 1003.1-200x extends historical practice by requiring that buffer execution be treated
16302 similarly. The reasons for this are two-fold. Historically, only the report by the last command
16303 executed from the buffer would be seen by the user, as each new report would overwrite the
16304 last. In addition, the standard developers believed that buffer execution had more in common
16305 with **global** and **v** commands than it did with other *ex* commands, and should behave similarly,
16306 for consistency and simplicity of specification.

showmatch, sm

The length of time the cursor spends on the matching character is unspecified because the timing capabilities of systems are often inexact and variable. The time should be long enough for the user to notice, but not long enough for the user to become annoyed. Some implementations of *vi* have added a **matchtime** option that permits users to set the number of 0,1 second intervals the cursor pauses on the matching character.

showmode

The **showmode** option has been used in some historical implementations of *ex* and *vi* to display the current editing mode when in open or visual mode. The editing modes have generally included ‘‘command’’ and ‘‘input’’, and sometimes other modes such as ‘‘replace’’ and ‘‘change’’. The string was usually displayed on the bottom line of the screen at the far right-hand corner. In addition, a preceding `'*'` character often denoted if the contents of the edit buffer had been modified. The latter display has sometimes been part of the **showmode** option, and sometimes based on another option. This option was not available in the 4 BSD historical implementation of *vi*, but was viewed as generally useful, particularly to novice users, and is required by IEEE Std 1003.1-200x.

The **smd** shorthand for the **showmode** option was not present in all historical implementations of the editor. IEEE Std 1003.1-200x requires it, for consistency.

Not all historical implementations of the editor displayed a mode string for command mode, differentiating command mode from text input mode by the absence of a mode string. IEEE Std 1003.1-200x permits this behavior for consistency with historical practice, but implementations are encouraged to provide a display string for both modes.

slowopen

Historically the **slowopen** option was automatically set if the terminal baud rate was less than 1 200 baud, or if the baud rate was 1 200 baud and the **redraw** option was not set. The **slowopen** option had two effects. First, when inserting characters in the middle of a line, characters after the cursor would not be pushed ahead, but would appear to be overwritten. Second, when creating a new line of text, lines after the current line would not be scrolled down, but would appear to be overwritten. In both cases, ending text input mode would cause the screen to be refreshed to match the actual contents of the edit buffer. Finally, terminals that were sufficiently intelligent caused the editor to ignore the **slowopen** option. IEEE Std 1003.1-200x permits most historical behavior, extending historical practice to require **slowopen** behaviors if the edit option is set by the user.

tags

The default path for tags files is left unspecified as implementations may have their own **tags** implementations that do not correspond to the historical ones. The default **tags** option value should probably at least include the file **./tags**.

16344 **term**

16345 Historical implementations of *ex* and *vi* ignored changes to the **term** edit option after the initial
16346 terminal information was loaded. This is permitted by IEEE Std 1003.1-200x; however,
16347 implementations are encouraged to permit the user to modify their terminal type at any time.

16348 **terse**

16349 Historically, the **terse** edit option optionally provided a shorter, less descriptive error message,
16350 for some error messages. This is permitted, but not required, by IEEE Std 1003.1-200x.
16351 Historically, most common visual mode errors (for example, trying to move the cursor past the
16352 end of a line) did not result in an error message, but simply alerted the terminal.
16353 Implementations wishing to provide messages for novice users are urged to do so based on the
16354 **edit** option **verbose**, and not **terse**.

16355 **window**

16356 In historical implementations, the default for the **window** edit option was based on the baud
16357 rate as follows:

16358 1. If the baud rate was less than 1 200, the **edit** option **w300** set the window value; for
16359 example, the line:

16360 `set w300=12`

16361 would set the window option to 12 if the baud rate was less than 1 200.

16362 2. If the baud rate was equal to 1 200, the **edit** option **w1200** set the window value.

16363 3. If the baud rate was greater than 1 200, the **edit** option **w9600** set the window value.

16364 The **w300**, **w1200**, and **w9600** options do not appear in IEEE Std 1003.1-200x because of their
16365 dependence on specific baud rates.

16366 In historical implementations, the size of the window displayed by various commands was
16367 related to, but not necessarily the same as, the **window** edit option. For example, the size of the
16368 window was set by the *ex* command **visual 10**, but it did not change the value of the **window**
16369 edit option. However, changing the value of the **window** edit option did change the number of
16370 lines that were displayed when the screen was repainted. IEEE Std 1003.1-200x does not permit
16371 this behavior in the interests of consistency and simplicity of specification, and requires that all
16372 commands that change the number of lines that are displayed do it by setting the value of the
16373 **window** edit option.

16374 **wrapmargin, wm**

16375 Historically, the **wrapmargin** option did not affect maps inserting characters that also had
16376 associated *count*s; for example **:map K 5aABC DEF**. Unfortunately, there are widely used
16377 maps that depend on this behavior. For consistency and simplicity of specification,
16378 IEEE Std 1003.1-200x does not permit this behavior.

16379 Historically, **wrapmargin** was calculated using the column display width of all characters on the
16380 screen. For example, an implementation using `"^I"` to represent <tab>s when the **list** edit
16381 option was set, where `'^'` and `'I'` each took up a single column on the screen, would calculate
16382 the **wrapmargin** based on a value of 2 for each <tab>. The **number** edit option similarly
16383 changed the effective length of the line as well. IEEE Std 1003.1-200x requires conformance to
16384 historical practice.

FUTURE DIRECTIONS

None.

SEE ALSO

ed, *sed*, *stty*, *vi*, the System Interfaces volume of IEEE Std 1003.1-200x, *access*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

The FUTURE DIRECTIONS section is added.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The obsolescent SYNOPSIS is removed, removing the **+***command* and – options.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the *map* command description, the sequence **#***digit* is added.
- The **directory**, **edcompatible**, **redraw**, and **slowopen** edit options are added.

The *ex* utility is extensively changed for alignment with the IEEE P1003.2b draft standard. This includes changes as a result of the IEEE PASC Interpretations 1003.2 #31, #38, #49, #50, #51, #52, #55, #56, #57, #61, #62, #63, #64, #65, and #78.

16403 **NAME**

16404 expand — convert tabs to spaces

16405 **SYNOPSIS**

16406 UP `expand [–t tablist][file ...]`

16407

16408 **DESCRIPTION**

16409 The *expand* utility shall write files or the standard input to the standard output with <tab>s
16410 replaced with one or more <space>s needed to pad to the next tab stop. Any <backspace>s shall
16411 be copied to the output and cause the column position count for tab stop calculations to be
16412 decremented; the column position count shall not be decremented below zero.

16413 **OPTIONS**

16414 The *expand* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
16415 12.2, Utility Syntax Guidelines.

16416 The following option shall be supported:

16417 **–t** *tablist* Specify the tab stops. The application shall ensure that the argument *tablist*
16418 consists of either a single positive decimal integer or a list of tabstops. If a single
16419 number is given, tabs shall be set that number of column positions apart instead of
16420 the default 8.

16421 If a list of tabstops is given, the application shall ensure that it consists of a list of
16422 two or more positive decimal integers, separated by <blank>s or commas, in
16423 ascending order. The tabs shall be set at those specific column positions. Each tab
16424 stop *N* shall be an integer value greater than zero, and the list is in strictly
16425 ascending order. This is taken to mean that, from the start of a line of output,
16426 tabbing to position *N* shall cause the next character output to be in the (*N*+1)th
16427 column position on that line.

16428 In the event of *expand* having to process a <tab> at a position beyond the last of
16429 those specified in a multiple tab-stop list, the <tab> shall be replaced by a single
16430 <space> in the output.

16431 **OPERANDS**

16432 The following operand shall be supported:

16433 *file* The pathname of a text file to be used as input.

16434 **STDIN**

16435 See the INPUT FILES section.

16436 **INPUT FILES**

16437 Input files shall be text files.

16438 **ENVIRONMENT VARIABLES**

16439 The following environment variables shall affect the execution of *expand*:

16440 *LANG* Provide a default value for the internationalization variables that are unset or null.
16441 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
16442 Internationalization Variables for the precedence of internationalization variables
16443 used to determine the values of locale categories.)

16444 *LC_ALL* If set to a non-empty string value, override the values of all the other
16445 internationalization variables.

16446 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
16447 characters (for example, single-byte as opposed to multi-byte 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 an output device.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
The standard output shall be equivalent to the input files with <tab>s converted into the appropriate number of <space>s.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion

>0 An error occurred.

CONSEQUENCES OF ERRORS
The *expand* utility shall terminate with an error message and non-zero exit status upon encountering difficulties accessing one of the *file* operands.

APPLICATION USAGE
None.

EXAMPLES
None.

RATIONALE
The *expand* utility is useful for preprocessing text files (before sorting, looking at specific columns, and so on) that contain <tab>s.

See the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.103, Column Position.

The *tablist* option-argument consists of integers in ascending order. Utility Syntax Guideline 8 mandates that *expand* shall accept the integers (within the single argument) separated using either commas or <blank>s.

FUTURE DIRECTIONS
None.

SEE ALSO
tabs, *unexpand*

16488 **CHANGE HISTORY**

16489 First released in Issue 4.

16490 **Issue 6**

16491 This utility is now marked as part of the User Portability Utilities option.

16492 The APPLICATION USAGE section is added.

16493 The obsolescent SYNOPSIS is removed.

16494 The *LC_CTYPE* environment variable description is updated to align with the IEEE P1003.2b
16495 draft standard.

16496 The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

16497 **NAME**

16498 expr — evaluate arguments as an expression

16499 **SYNOPSIS**

16500 `expr` *operand*

16501 **DESCRIPTION**

16502 The *expr* utility shall evaluate an expression and write the result to standard output.

16503 **OPTIONS**

16504 None.

16505 **OPERANDS**

16506 The single expression evaluated by *expr* shall be formed from the operands, as described in the
16507 EXTENDED DESCRIPTION section. The application shall ensure that each of the expression
16508 operator symbols:

16509 `( ) | & = > >= < <= != + – * / % :`

16510 and the symbols *integer* and *string* in the table are provided as separate arguments to *expr*.

16511 **STDIN**

16512 Not used.

16513 **INPUT FILES**

16514 None.

16515 **ENVIRONMENT VARIABLES**

16516 The following environment variables shall affect the execution of *expr*:

16517 *LANG* Provide a default value for the internationalization variables that are unset or null.
16518 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
16519 Internationalization Variables for the precedence of internationalization variables
16520 used to determine the values of locale categories.)

16521 *LC_ALL* If set to a non-empty string value, override the values of all the other
16522 internationalization variables.

16523 *LC_COLLATE*
16524 Determine the locale for the behavior of ranges, equivalence classes, and multi-
16525 character collating elements within regular expressions and by the string
16526 comparison operators.

16527 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
16528 characters (for example, single-byte as opposed to multi-byte characters in
16529 arguments) and the behavior of character classes within regular expressions.

16530 *LC_MESSAGES*
16531 Determine the locale that should be used to affect the format and contents of
16532 diagnostic messages written to standard error.

16533 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

16534 **ASYNCHRONOUS EVENTS**

16535 Default.

16536 **STDOUT**

16537 The *expr* utility shall evaluate the expression and write the result, followed by a <newline>, to
16538 standard output.

16539 **STDERR**

16540 The standard error shall be used only for diagnostic messages.

16541 **OUTPUT FILES**

16542 None.

16543 **EXTENDED DESCRIPTION**

16544 The formation of the expression to be evaluated is shown in the following table. The symbols
16545 *expr*, *expr1*, and *expr2* represent expressions formed from *integer* and *string* symbols and the
16546 expression operator symbols (all separate arguments) by recursive application of the constructs
16547 described in the table. The expressions are listed in order of increasing precedence, with equal-
16548 precedence operators grouped between horizontal lines. All of the operators shall be left-
16549 associative.

16550

Expression	Description
expr1 | *expr2*	Returns the evaluation of *expr1* if it is neither null nor zero; otherwise, returns the evaluation of *expr2* if it is not null; otherwise, zero.
expr1 & *expr2*	Returns the evaluation of *expr1* if neither expression evaluates to null or zero; otherwise, returns zero.
	Returns the result of a decimal integer comparison if both arguments are integers; otherwise, returns the result of a string comparison using the locale-specific collation sequence. The result of each comparison is 1 if the specified relationship is true, or 0 if the relationship is false.
expr1 = *expr2*	Equal.
expr1 > *expr2*	Greater than.
expr1 >= *expr2*	Greater than or equal.
expr1 < *expr2*	Less than.
expr1 <= *expr2*	Less than or equal.
expr1 != *expr2*	Not equal.
expr1 + *expr2*	Addition of decimal integer-valued arguments.
expr1 – *expr2*	Subtraction of decimal integer-valued arguments.
expr1 * *expr2*	Multiplication of decimal integer-valued arguments.
expr1 / *expr2*	Integer division of decimal integer-valued arguments, producing an integer result.
expr1 % *expr2*	Remainder of integer division of decimal integer-valued arguments.
expr1 : *expr2*	Matching expression; see below.
(*expr*)	Grouping symbols. Any expression can be placed within parentheses. Parentheses can be nested to a depth of {EXPR_NEST_MAX}.
integer	An argument consisting only of an (optional) unary minus followed by digits.
string	A string argument; see below.

16581 **Matching Expression**

16582 The `':'` matching operator shall compare the string resulting from the evaluation of *expr1* with
16583 the regular expression pattern resulting from the evaluation of *expr2*. Regular expression syntax
16584 shall be that defined in the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.3, Basic
16585 Regular Expressions, except that all patterns are anchored to the beginning of the string (that is,
16586 only sequences starting at the first character of a string are matched by the regular expression)
16587 and, therefore, it is unspecified whether `'^'` is a special character in that context. Usually, the
16588 matching operator shall return a string representing the number of characters matched (`'0'` on
16589 failure). Alternatively, if the pattern contains at least one regular expression subexpression
16590 `"[\(...\)]"`, the string corresponding to `"\1"` shall be returned.

16591 **String Operand**

16592 A string argument is an argument that cannot be identified as an *integer* argument or as one of
16593 the expression operator symbols shown in the OPERANDS section.

16594 The use of string arguments **length**, **substr**, **index**, or **match** produces unspecified results.

16595 **EXIT STATUS**

16596 The following exit values shall be returned:

16597 0 The *expression* evaluates to neither null nor zero.

16598 1 The *expression* evaluates to null or zero.

16599 2 Invalid *expression*.

16600 >2 An error occurred.

16601 **CONSEQUENCES OF ERRORS**

16602 Default.

16603 **APPLICATION USAGE**

16604 After argument processing by the shell, *expr* is not required to be able to tell the difference
16605 between an operator and an operand except by the value. If `"$a"` is `'='`, the command:

16606 ```
expr $a = '='
```

16607 looks like:

16608 ```
expr = = =
```

16609 as the arguments are passed to *expr* (and they all may be taken as the `'='` operator). The
16610 following works reliably:

16611 ```
expr X$a = X=
```

16612 Also note that this volume of IEEE Std 1003.1-200x permits implementations to extend utilities.
16613 The *expr* utility permits the integer arguments to be preceded with a unary minus. This means
16614 that an integer argument could look like an option. Therefore, the conforming application must |
16615 employ the `"--"` construct of Guideline 10 of the Base Definitions volume of |
16616 IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines to protect its operands if there is
16617 any chance the first operand might be a negative integer (or any string with a leading minus).

16618 **EXAMPLES**

16619 The *expr* utility has a rather difficult syntax:

16620 • Many of the operators are also shell control operators or reserved words, so they have to be
16621 escaped on the command line.
```

- Each part of the expression is composed of separate arguments, so liberal usage of <blank>s is required. For example:

Invalid	Valid
`expr 1+2`	`expr 1 + 2`
`expr "1 + 2"`	`expr 1 + 2`
`expr 1 + (2 * 3)`	`expr 1 + \( 2 \* 3 \)`

In many cases, the arithmetic and string features provided as part of the shell command language are easier to use than their equivalents in *expr*. Newly written scripts should avoid *expr* in favor of the new features within the shell; see Section 2.5 (on page 2235) and Section 2.6.4 (on page 2243).

The following command:

```
a=$(expr $a + 1)
```

adds 1 to the variable *a*.

The following command, for `"$a"` equal to either **/usr/abc/file** or just **file**:

```
expr $a : '.*/\(.*\)' \| $a
```

returns the last segment of a pathname (that is, **file**). Applications should avoid the character `'/'` used alone as an argument: *expr* may interpret it as the division operator.

The following command:

```
expr "//$a" : '.*/\(.*\)'
```

is a better representation of the previous example. The addition of the `"//"` characters eliminates any ambiguity about the division operator and simplifies the whole expression. Also note that pathnames may contain characters contained in the *IFS* variable and should be quoted to avoid having `"$a"` expand into multiple arguments.

The following command:

```
expr "$VAR" : '.*'
```

returns the number of characters in *VAR*.

RATIONALE

In an early proposal, EREs were used in the matching expression syntax. This was changed to BREs to avoid breaking historical applications.

The use of a leading circumflex in the BRE is unspecified because many historical implementations have treated it as a special character, despite their system documentation. For example:

```
expr foo : ^foo      expr ^foo : ^foo
```

return 3 and 0, respectively, on those systems; their documentation would imply the reverse. Thus, the anchoring condition is left unspecified to avoid breaking historical scripts relying on this undocumented feature.

FUTURE DIRECTIONS

None.

16661 **SEE ALSO**

16662 Section 2.6.4

16663 **CHANGE HISTORY**

16664 First released in Issue 2.

16665 **Issue 5**

16666 FUTURE DIRECTIONS section added.

16667 **Issue 6**

16668 The *expr* utility is aligned with the IEEE P1003.2b draft standard, to include resolution of IEEE
16669 PASC Interpretation 1003.2 #104.

16670 The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

16671 **NAME**

16672 false — return false value

16673 **SYNOPSIS**

16674 `false`

16675 **DESCRIPTION**

16676 The *false* utility shall return with a non-zero exit code.

16677 **OPTIONS**

16678 None.

16679 **OPERANDS**

16680 None.

16681 **STDIN**

16682 Not used.

16683 **INPUT FILES**

16684 None.

16685 **ENVIRONMENT VARIABLES**

16686 None.

16687 **ASYNCHRONOUS EVENTS**

16688 Default.

16689 **STDOUT**

16690 Not used.

16691 **STDERR**

16692 None.

16693 **OUTPUT FILES**

16694 None.

16695 **EXTENDED DESCRIPTION**

16696 None.

16697 **EXIT STATUS**

16698 The *false* utility always shall exit with a value other than zero.

16699 **CONSEQUENCES OF ERRORS**

16700 Default.

16701 **APPLICATION USAGE**

16702 None.

16703 **EXAMPLES**

16704 None.

16705 **RATIONALE**

16706 None.

16707 **FUTURE DIRECTIONS**

16708 None.

16709 **SEE ALSO**

16710 *true*

16711 **CHANGE HISTORY**
16712 First released in Issue 2.

16713 **NAME**

16714 fc — process the command history list

16715 **SYNOPSIS**

16716 UP
```
fc [–r][–e editor] [first[last]]
```

16717
```
fc –l[–nr] [first[last]]
```

16718
```
fc –s[old=new][first]
```

16719

16720 **DESCRIPTION**

16721 The *fc* utility shall list, or shall edit and re-execute, commands previously entered to an
16722 interactive *sh*.

16723 The command history list shall reference commands by number. The first number in the list is
16724 selected arbitrarily. The relationship of a number to its command shall not change except when
16725 the user logs in and no other process is accessing the list, at which time the system may reset the
16726 numbering to start the oldest retained command at another number (usually 1). When the
16727 number reaches an implementation-defined upper limit, which shall be no smaller than the
16728 value in *HISTSIZE* or 32 767 (whichever is greater), the shell may wrap the numbers, starting the
16729 next command with a lower number (usually 1). However, despite this optional wrapping of
16730 numbers, *fc* shall maintain the time-ordering sequence of the commands. For example, if four
16731 commands in sequence are given the numbers 32 766, 32 767, 1 (wrapped), and 2 as they are
16732 executed, command 32 767 is considered the command previous to 1, even though its number is
16733 higher.

16734 When commands are edited (when the **–l** option is not specified), the resulting lines shall be
16735 entered at the end of the history list and then re-executed by *sh*. The *fc* command that caused the
16736 editing shall not be entered into the history list. If the editor returns a non-zero exit status, this
16737 shall suppress the entry into the history list and the command re-execution. Any command line
16738 variable assignments or redirection operators used with *fc* shall affect both the *fc* command itself
16739 as well as the command that results; for example:

16740
```
fc –s –– –1 2>/dev/null
```

16741 reinvokes the previous command, suppressing standard error for both *fc* and the previous
16742 command.

16743 **OPTIONS**

16744 The *fc* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
16745 Utility Syntax Guidelines.

16746 The following options shall be supported:

16747 **–e** *editor* Use the editor named by *editor* to edit the commands. The *editor* string is a utility
16748 name, subject to search via the *PATH* variable (see the Base Definitions volume of
16749 IEEE Std 1003.1-200x, Chapter 8, Environment Variables). The value in the *FCEDIT*
16750 variable shall be used as a default when **–e** is not specified. If *FCEDIT* is null or
16751 unset, *ed* shall be used as the editor.

16752 **–l** (The letter ell.) List the commands rather than invoking an editor on them. The
16753 commands shall be written in the sequence indicated by the *first* and *last* operands,
16754 as affected by **–r**, with each command preceded by the command number.

16755 **–n** Suppress command numbers when listing with **–l**.

16756 **–r** Reverse the order of the commands listed (with **–l**) or edited (with neither **–l** nor
16757 **–s**).

16758 **–s** Reexecute the command without invoking an editor.

16759 **OPERANDS**

16760 The following operands shall be supported:

16761 *first*, *last*

16762 Select the commands to list or edit. The number of previous commands that can be
16763 accessed shall be determined by the value of the *HISTSIZE* variable. The value of
16764 *first* or *last* or both shall be one of the following:

16765 **[+]***number* A positive number representing a command number; command
16766 numbers can be displayed with the **–l** option.

16767 *–number* A negative decimal number representing the command that was
16768 executed *number* of commands previously. For example, –1 is the
16769 immediately previous command.

16770 *string* A string indicating the most recently entered command that begins
16771 with that string. If the *old=new* operand is not also specified with **–s**,
16772 the string form of the *first* operand cannot contain an embedded
16773 equal sign.

16774 When the synopsis form with **–s** is used:

16775 • If *first* is omitted, the previous command shall be used.

16776 For the synopsis forms without **–s**:

16777 • If *last* is omitted, *last* shall default to the previous command when **–l** is
16778 specified; otherwise, it shall default to *first*.

16779 • If *first* and *last* are both omitted, the previous 16 commands shall be listed or
16780 the previous single command shall be edited (based on the **–l** option).

16781 • If *first* and *last* are both present, all of the commands from *first* to *last* shall be
16782 edited (without **–l**) or listed (with **–l**). Editing multiple commands shall be
16783 accomplished by presenting to the editor all of the commands at one time, each
16784 command starting on a new line. If *first* represents a newer command than *last*,
16785 the commands shall be listed or edited in reverse sequence, equivalent to using
16786 **–r**. For example, the following commands on the first line are equivalent to the
16787 corresponding commands on the second:

16788 `fc –r 10 20    fc    30 40`
16789 `fc    20 10    fc –r 40 30`

16790 • When a range of commands is used, it shall not be an error to specify *first* or *last*
16791 values that are not in the history list; *fc* shall substitute the value representing
16792 the oldest or newest command in the list, as appropriate. For example, if there
16793 are only ten commands in the history list, numbered 1 to 10:

16794 `fc –l`
16795 `fc 1 99`

16796 shall list and edit, respectively, all ten commands.

16797 *old=new* Replace the first occurrence of string *old* in the commands to be re-executed by the
16798 string *new*.

16799 **STDIN**

16800 Not used.

16801 **INPUT FILES**

16802 None.

16803 **ENVIRONMENT VARIABLES**

16804 The following environment variables shall affect the execution of *fc*:

16805 *FCEDIT* This variable, when expanded by the shell, shall determine the default value for
16806 the **–e** *editor* option's *editor* option-argument. If *FCEDIT* is null or unset, *ed* shall be
16807 used as the editor.

16808 *HISTFILE* Determine a pathname naming a command history file. If the *HISTFILE* variable is
16809 not set, the shell may attempt to access or create a file **.sh_history** in the directory
16810 referred to by the *HOME* environment variable. If the shell cannot obtain both read
16811 and write access to, or create, the history file, it shall use an unspecified
16812 mechanism that allows the history to operate properly. (References to history
16813 ''file'' in this section shall be understood to mean this unspecified mechanism in
16814 such cases.) An implementation may choose to access this variable only when
16815 initializing the history file; this initialization shall occur when *fc* or *sh* first attempt
16816 to retrieve entries from, or add entries to, the file, as the result of commands issued
16817 by the user, the file named by the *ENV* variable, or implementation-defined system
16818 start-up files. In some historical shells, the history file is initialized just after the
16819 *ENV* file has been processed. Therefore, it is implementation-defined whether
16820 changes made to *HISTFILE* after the history file has been initialized are effective.
16821 Implementations may choose to disable the history list mechanism for users with
16822 appropriate privileges who do not set *HISTFILE*; the specific circumstances under
16823 which this occurs are implementation-defined. If more than one instance of the
16824 shell is using the same history file, it is unspecified how updates to the history file
16825 from those shells interact. As entries are deleted from the history file, they shall be
16826 deleted oldest first. It is unspecified when history file entries are physically
16827 removed from the history file.

16828 *HISTSIZE* Determine a decimal number representing the limit to the number of previous
16829 commands that are accessible. If this variable is unset, an unspecified default
16830 greater than or equal to 128 shall be used. The maximum number of commands in
16831 the history list is unspecified, but shall be at least 128. An implementation may
16832 choose to access this variable only when initializing the history file, as described
16833 under *HISTFILE*. Therefore, it is unspecified whether changes made to *HISTSIZE*
16834 after the history file has been initialized are effective.

16835 *LANG* Provide a default value for the internationalization variables that are unset or null.
16836 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
16837 Internationalization Variables for the precedence of internationalization variables
16838 used to determine the values of locale categories.)

16839 *LC_ALL* If set to a non-empty string value, override the values of all the other
16840 internationalization variables.

16841 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
16842 characters (for example, single-byte as opposed to multi-byte characters in
16843 arguments and input files).

16844 *LC_MESSAGES*

16845 Determine the locale that should be used to affect the format and contents of
16846 diagnostic messages written to standard error.

16847 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

16848 **ASYNCHRONOUS EVENTS**
16849 Default.

16850 **STDOUT**
16851 When the **–l** option is used to list commands, the format of each command in the list shall be as
16852 follows:

16853 `"%d\t%s\n", <line number>, <command>`

16854 If both the **–l** and **–n** options are specified, the format of each command shall be:

16855 `"\t%s\n", <command>`

16856 If the <*command*> consists of more than one line, the lines after the first shall be displayed as:

16857 `"\t%s\n", <continued-command>`

16858 **STDERR**
16859 The standard error shall be used only for diagnostic messages.

16860 **OUTPUT FILES**
16861 None.

16862 **EXTENDED DESCRIPTION**
16863 None.

16864 **EXIT STATUS**
16865 The following exit values shall be returned:

16866 0 Successful completion of the listing.

16867 >0 An error occurred.

16868 Otherwise, the exit status shall be that of the commands executed by *fc*.

16869 **CONSEQUENCES OF ERRORS**
16870 Default.

16871 **APPLICATION USAGE**
16872 Since editors sometimes use file descriptors as integral parts of their editing, redirecting their file
16873 descriptors as part of the *fc* command can produce unexpected results. For example, if *vi* is the
16874 *FCEDIT* editor, the command:

16875 `fc –s | more`

16876 does not work correctly on many systems.

16877 Users on windowing systems may want to have separate history files for each window by
16878 setting *HISTFILE* as follows:

16879 `HISTFILE=$HOME/.sh_hist$$`

16880 **EXAMPLES**
16881 None.

16882 **RATIONALE**
16883 This utility is based on the *fc* built-in of the KornShell.

16884 An early proposal specified the **–e** option as **[–e** *editor* **[***old***=** *new* **]]**, which is not historical
16885 practice. Historical practice in *fc* of either **[–e** *editor***]** or **[–e** – **[** *old***=** *new* **]]** is acceptable, but not
16886 both together. To clarify this, a new option **–s** was introduced replacing the **[–e –]**. This resolves
16887 the conflict and makes *fc* conform to the Utility Syntax Guidelines.

HISTFILE Some implementations of the KornShell check for the superuser 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 superuser to have history.

HISTSIZE Needed to limit the size of history files. It is the intent of the standard developers 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.

The initialization process for the history file can be dependent on the system start-up files, in that they may contain commands that effectively preempt the settings the user has for *HISTFILE* and *HISTSIZE*. For example, function definition commands are recorded in the history file. If the system administrator includes function definitions in some system start-up file called before the *ENV* file, the history file is initialized before the user can influence its characteristics. In some historical shells, the history file is initialized just after the *ENV* file has been processed. Because of these situations, the text requires the initialization process to be implementation-defined.

Consideration was given to omitting the *fc* utility in favor of the command line editing feature in *sh*. For example, in *vi* editing mode, typing "<ESC> v" is equivalent to:

```
EDITOR=vi fc
```

However, the *fc* utility allows the user the flexibility to edit multiple commands simultaneously (such as *fc* 10 20) and to use editors other than those supported by *sh* for command line editing.

In the KornShell, the alias **r** (''re-do'') is preset to *fc* –**e** – (equivalent to the POSIX *fc* –**s**). This is probably an easier command name to remember than *fc* (''fix command''), but it does not meet the Utility Syntax Guidelines. Renaming *fc* to *hist* or *redo* was considered, but since this description closely matches historical KornShell practice already, such a renaming was seen as gratuitous. Users are free to create aliases whenever odd historical names such as *fc*, *awk*, *cat*, *grep*, or *yacc* are standardized by POSIX.

Command numbers have no ordering effects; they are like serial numbers. The –**r** option and –*number* operand address the sequence of command execution, regardless of serial numbers. So, for example, if the command number wrapped back to 1 at some arbitrary point, there would be no ambiguity associated with traversing the wrap point. For example, if the command history were:

```
32766: echo 1
32767: echo 2
1: echo 3
```

the number –2 refers to command 32 767 because it is the second previous command, regardless of serial number.

FUTURE DIRECTIONS

None.

SEE ALSO

sh

CHANGE HISTORY

First released in Issue 4.

Issue 5

FUTURE DIRECTIONS section added.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

In the ENVIRONMENT VARIABLES section, the text ‘‘user’s home directory’’ is updated to ‘‘directory referred to by the *HOME* environment variable’’.

16936 **NAME**

16937 fg — run jobs in the foreground

16938 **SYNOPSIS**

16939 UP `fg [job_id]`

16940

16941 **DESCRIPTION**

16942 If job control is enabled (see the description of *set* –**m**), the *fg* utility shall move a background job
16943 from the current environment (see Section 2.12 (on page 2263)) into the foreground.

16944 Using *fg* to place a job into the foreground shall remove its process ID from the list of those
16945 ‘‘known in the current shell execution environment’’; see Section 2.9.3.1 (on page 2252).

16946 **OPTIONS**

16947 None.

16948 **OPERANDS**

16949 The following operand shall be supported:

16950 *job_id* Specify the job to be run as a foreground job. If no *job_id* operand is given, the
16951 *job_id* for the job that was most recently suspended, placed in the background or
16952 run as a background job, shall be used. The format of *job_id* is described in the Base
16953 Definitions volume of IEEE Std 1003.1-200x, Section 3.203, Job Control Job ID.

16954 **STDIN**

16955 Not used.

16956 **INPUT FILES**

16957 None.

16958 **ENVIRONMENT VARIABLES**

16959 The following environment variables shall affect the execution of *fg*:

16960 *LANG* Provide a default value for the internationalization variables that are unset or null.
16961 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
16962 Internationalization Variables for the precedence of internationalization variables
16963 used to determine the values of locale categories.)

16964 *LC_ALL* If set to a non-empty string value, override the values of all the other
16965 internationalization variables.

16966 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
16967 characters (for example, single-byte as opposed to multi-byte characters in
16968 arguments).

16969 *LC_MESSAGES*
16970 Determine the locale that should be used to affect the format and contents of
16971 diagnostic messages written to standard error.

16972 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

16973 **ASYNCHRONOUS EVENTS**

16974 Default.

16975 **STDOUT**

16976 The *fg* utility shall write the command line of the job to standard output in the following format:

16977 `"%s\n", <command>`

16978 **STDERR**

16979 The standard error shall be used only for diagnostic messages.

16980 **OUTPUT FILES**

16981 None.

16982 **EXTENDED DESCRIPTION**

16983 None.

16984 **EXIT STATUS**

16985 The following exit values shall be returned:

16986 0 Successful completion.

16987 >0 An error occurred.

16988 **CONSEQUENCES OF ERRORS**

16989 If job control is disabled, the *fg* utility shall exit with an error and no job shall be placed in the
16990 foreground.

16991 **APPLICATION USAGE**

16992 The *fg* utility does not work as expected when it is operating in its own utility execution
16993 environment because that environment has no applicable jobs to manipulate. See the
16994 APPLICATION USAGE section for *bg* (on page 2410). For this reason, *fg* is generally
16995 implemented as a shell regular built-in.

16996 **EXAMPLES**

16997 None.

16998 **RATIONALE**

16999 The extensions to the shell specified in this volume of IEEE Std 1003.1-200x have mostly been
17000 based on features provided by the KornShell. The job control features provided by *bg*, *fg*, and *jobs*
17001 are also based on the KornShell. The standard developers examined the characteristics of the C
17002 shell versions of these utilities and found that differences exist. Despite widespread use of the C
17003 shell, the KornShell versions were selected for this volume of IEEE Std 1003.1-200x to maintain a
17004 degree of uniformity with the rest of the KornShell features selected (such as the very popular
17005 command line editing features).

17006 **FUTURE DIRECTIONS**

17007 None.

17008 **SEE ALSO**

17009 *bg*, *kill*, *jobs*, *wait*

17010 **CHANGE HISTORY**

17011 First released in Issue 4.

17012 **Issue 6**

17013 This utility is now marked as part of the User Portability Utilities option.

17014 The APPLICATION USAGE section is added.

17015 The JC marking is removed from the SYNOPSIS since job control is mandatory is this issue.

17016 **NAME**

17017 file — determine file type

17018 **SYNOPSIS**

17019 UP
```
file [–dhi][–M file][–m file] file ...
```
17020

17021 **DESCRIPTION**

17022 The *file* utility shall perform a series of tests on each specified *file* in an attempt to classify it:

17023 1. If the file is not a regular file, its file type shall be identified. The file types directory, FIFO, socket, block special, and character special shall be identified as such. Other implementation-defined file types may also be identified.

17026 2. If the file is a regular file, and:

17027 a. The file is zero-length, it shall be identified as an empty file.

17028 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.)

17031 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.

17033 If *file* is a symbolic link, by default the link shall be resolved and *file* shall test the type of file referenced by the symbolic link.

17035 **OPTIONS**

17036 The *file* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

17038 The following options shall be supported by the implementation:

17039 **–d** Apply any default system tests to the file.

17040 **–h** When a symbolic link is encountered, identify the file as a symbolic link. If **–h** is not specified and *file* is a symbolic link that refers to a nonexistent file, *file* shall identify the file as a symbolic link, as if **–h** had been specified.

17043 **–i** If a file is a regular file, do not attempt to classify the type of the file further, but identify the file as specified in the STDOUT section, using a <*type*> string that contains the string `"regular file"`.

17046 **–M** *file* Specify the name of a file containing tests that shall be applied to a file in order to classify it (see the EXTENDED DESCRIPTION). No default system tests shall be applied.

17049 **–m** *file* Specify the name of a file containing tests that shall be applied to a file in order to classify it (see the EXTENDED DESCRIPTION).

17051 If multiple instances of the **–m**, **–d**, or **–M** options are specified, the concatenation of the tests specified, in the order specified, shall be the set of tests that are applied. If a **–M** option is specified, no tests other than those specified using the **–d**, **–M**, and **–m** options shall be applied to the file. If neither the **–d** nor **–M** options are specified, any default system tests shall be applied after any tests specified using the **–m** option.

17056 **OPERANDS**

17057 The following operand shall be supported:

17058 *file* A pathname of a file to be tested.

17059 **STDIN**

17060 Not used.

17061 **INPUT FILES**

17062 The *file* can be any file type.

17063 **ENVIRONMENT VARIABLES**

17064 The following environment variables shall affect the execution of *file*:

17065 *LANG* Provide a default value for the internationalization variables that are unset or null.
17066 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
17067 Internationalization Variables for the precedence of internationalization variables
17068 used to determine the values of locale categories.)

17069 *LC_ALL* If set to a non-empty string value, override the values of all the other
17070 internationalization variables.

17071 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
17072 characters (for example, single-byte as opposed to multi-byte characters in
17073 arguments and input files).

17074 *LC_MESSAGES*
17075 Determine the locale that should be used to affect the format and contents of
17076 diagnostic messages written to standard error and informative messages written to
17077 standard output.

17078 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

17079 **ASYNCHRONOUS EVENTS**

17080 Default.

17081 **STDOUT**

17082 In the POSIX locale, the following format shall be used to identify each operand, *file* specified:

17083 `"%s: %s\n", <file>, <type>`

17084 The values for <*type*> are unspecified, except that in the POSIX locale, if *file* is identified as one
17085 of the types listed in the following table, <*type*> shall contain (but is not limited to) the
17086 corresponding string. Each space shown in the strings shall be exactly one <space>.

17087 **Table 4-8** File Utility Output Strings

17088 **If *file* is a:**	**<*type*> shall contain the string:**
17089 Directory	directory
17090 FIFO	fifo
17091 Socket	socket
17092 Block special	block special
17093 Character special	character special
17094 Executable binary	executable
17095 Empty regular file	empty
17096 Symbolic link	symbolic link to
17097 *ar* archive library (see *ar*)	archive
17098 Extended *cpio* format (see *pax*)	cpio archive
17099 Extended *tar* format (see **ustar** in *pax*)	tar archive
17100 Shell script	commands text
17101 C-language source	c program text
17102 FORTRAN source	fortran program text

17103 If *file* is identified as a symbolic link (see **–h**), the following alternative output format shall be
17104 used:

17105 `"%s: %s %s\n", <file>, <type>, <contents of link>"`

17106 If the file named by the *file* operand does not exist or cannot be read, the string `"cannot open"`
17107 shall be included as part of the <*type*> field, but this shall not be considered an error that affects
17108 the exit status. If the type of the file named by the *file* operand cannot be determined, the string
17109 `"data"` shall be included as part of the <*type*> field, but this shall not be considered an error that
17110 affects the exit status.

17111 **STDERR**

17112 The standard error shall be used only for diagnostic messages.

17113 **OUTPUT FILES**

17114 None.

17115 **EXTENDED DESCRIPTION**

17116 A file specified as an option-argument to the **–m** or **–M** options shall contain one test per line,
17117 which shall be applied to the file. If the test succeeds, the message field of the line shall be
17118 printed and no further tests shall be applied, with the exception that tests on immediately
17119 following lines beginning with a single `'>'` character shall be applied.

17120 Each line shall be composed of the following four <blank>-separated fields:

17121 *offset* An unsigned number (optionally preceded by a single `'>'` character) specifying
17122 the *offset*, in bytes, of the value in the file that is to be compared against the *value*
17123 field of the line. If the file is shorter than the specified offset, the test shall fail.

17124 If the *offset* begins with the character `'>'`, the test contained in the line shall not be
17125 applied to the file unless the test on the last line for which the *offset* did not begin
17126 with a `'>'` was successful. By default, the *offset* shall be interpreted as an unsigned
17127 decimal number. With a leading 0x or 0X, the *offset* shall be interpreted as a
17128 hexadecimal number; otherwise, with a leading 0, the *offset* shall be interpreted as
17129 an octal number.

17130 *type* The type of the value in the file to be tested. The type shall consist of the type
17131 specification characters `c`, `d`, `f`, `s`, and `u`, specifying character, signed decimal,
17132 floating point, string, and unsigned decimal, respectively.

The *type* string shall be interpreted as the bytes from the file starting at the specified *offset* and including the same number of bytes specified by the *value* field. If insufficient bytes remain in the file past the *offset* to match the *value* field, the test shall fail.

The type specification characters `d`, `f`, and `u` can be followed by an optional unsigned decimal integer that specifies the number of bytes represented by the type. The type specification character `f` can be followed by an optional `F`, `D`, or `L`, indicating that the value is of type **float**, **double**, or **long double**, respectively. The type specification characters `d` and `u` can be followed by an optional `C`, `S`, `I`, or `L`, indicating that the value is of type **char**, **short**, **int**, or **long**, respectively.

The default number of bytes represented by the type specifiers `d`, `f`, and `u` shall correspond to their respective C-language types as follows. If the system claims conformance to the C-Language Development Utilities option, those specifiers shall correspond to the default sizes used in the *c99* utility. Otherwise, the default sizes shall be implementation-defined.

For the type specifier characters `d` and `u`, the default number of bytes shall correspond to the size of a basic integer type of the implementation. For these specifier characters, the implementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-language types **char**, **short**, **int**, or **long**. These numbers can also be specified by an application as the characters `C`, `S`, `I`, and `L`, respectively. The byte order used when interpreting numeric values is implementation-defined, but shall correspond to the order in which a constant of the corresponding type is stored in memory on the system.

For the type specifier `f`, the default number of bytes shall correspond to the number of bytes in the basic double precision floating-point data type of the underlying implementation. The implementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-language types **float**, **double**, and **long double**. These numbers can also be specified by an application as the characters `F`, `D`, and `L`, respectively.

All type specifiers, except for `s`, can be followed by a mask specifier of the form &*number*. The mask value shall be AND'ed with the value before the comparison with the value from the file is made. By default, the mask shall be interpreted as an unsigned decimal number. With a leading 0x or 0X, the mask shall be interpreted as an unsigned hexadecimal number; otherwise, with a leading 0, the mask shall be interpreted as an unsigned octal number.

The strings **byte**, **short**, **long**, and **string** shall also be supported as type fields, being interpreted as `dC`, `dS`, `dL`, and `s`, respectively.

value The *value* to be compared with the value from the file.

If the specifier from the type field is `s` or **string**, then interpret the value as a string. Otherwise, interpret it as a number. If the value is a string, then the test shall succeed only when a string value exactly matches the bytes from the file.

If the *value* is a string, it can contain the following sequences:

`\`*character* The backslash-escape sequences as specified in the Base Definitions volume of IEEE Std 1003.1-200x, Table 5-1, Escape Sequences and Associated Actions (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`). The results of using any other

17180 character, other than an octal digit, following the backslash are
17181 unspecified.

17182 *\octal* Octal sequences that can be used to represent characters with
17183 specific coded values. An octal sequence shall consist of a
17184 backslash followed by the longest sequence of one, two, or three
17185 octal-digit characters (01234567). If the size of a byte on the
17186 system is greater than 9 bits, the valid escape sequence used to
17187 represent a byte is implementation-defined.

17188 By default, any value that is not a string shall be interpreted as a signed decimal
17189 number. Any such value, with a leading 0x or 0X, shall be interpreted as an
17190 unsigned hexadecimal number; otherwise, with a leading zero, the value shall be
17191 interpreted as an unsigned octal number.

17192 If the value is not a string, it can be preceded by a character indicating the
17193 comparison to be performed. Permissible characters and the comparisons they
17194 specify are as follows:

17195 = The test shall succeed if the value from the file equals the *value* field.

17196 < The test shall succeed if the value from the file is less than the *value* field.

17197 > The test shall succeed if the value from the file is greater than the *value* field.

17198 & The test shall succeed if all of the bits in the *value* field are set in the value
17199 from the file.

17200 ^ The test shall succeed if at least one of the bits in the *value* field is not set in the
17201 value from the file.

17202 x The test shall succeed if the file is large enough to contain a value of the type
17203 specified starting at the offset specified.

17204 *message* The *message* to be printed if the test succeeds. The *message* shall be interpreted
17205 using the notation for the *printf* formatting specification; see *printf*. If the *value*
17206 field was a string, then the value from the file shall be the argument for the *printf*
17207 formatting specification; otherwise, the value from the file shall be the argument.

17208 **EXIT STATUS**

17209 The following exit values shall be returned:

17210 0 Successful completion.

17211 >0 An error occurred.

17212 **CONSEQUENCES OF ERRORS**

17213 Default.

17214 **APPLICATION USAGE**

17215 The *file* utility can only be required to guess at many of the file types because only exhaustive
17216 testing can determine some types with certainty. For example, binary data on some
17217 implementations might match the initial segment of an executable or a *tar* archive.

17218 Note that the table indicates that the output contains the stated string. Systems may add text
17219 before or after the string. For executables, as an example, the machine architecture and various
17220 facts about how the file was link-edited may be included.

EXAMPLES

Determine whether an argument is a binary executable file:

```
file "$1" | grep –Fq executable &&
    printf "%s is executable.\n" "$1"
```

RATIONALE

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 utilities are listed in the table.

Implementations that support symbolic links are encouraged to use the string `"symbolic link"` to identify them.

Historical systems have used a ‘‘magic file’’ named **/etc/magic** to help identify file types. Because it is generally useful for users and scripts to be able to identify special file types, the **–m** flag and a portable format for user-created magic files has been specified. No requirement is made that an implementation of *file* use this method of identifying files, only that users be permitted to add their own classifying tests.

In addition, three options have been added to historical practice. The **–d** flag has been added to permit users to cause their tests to follow any default system tests. The **–i** flag has been added to permit users to test portably for regular files in shell scripts. The **–M** flag has been added to permit users to ignore any default system tests.

The historical **–c** option was omitted as not particularly useful to users or portable shell scripts. In addition, a reasonable implementation of the *file* utility would report any errors found each time the magic file is read.

The historical format of the magic file was the same as that specified by the Rationale in the previous version of IEEE Std 1003.1-200x for the *offset*, *value*, and *message* fields; however, it used less precise type fields than the format specified by the current normative text. The new type field values are a superset of the historical ones.

The following is an example magic file:

```
0  short     070707              cpio archive
0  short     0143561             Byte-swapped cpio archive
0  string    070707              ASCII cpio archive
0  long      0177555             Very old archive
0  short     0177545             Old archive
0  short     017437              Old packed data
0  string    \037\036            Packed data
0  string    \377\037            Compacted data
0  string    \037\235            Compressed data
>2 byte&0x80 >0                  Block compressed
>2 byte&0x1f x                   %d bits
0  string    \032\001            Compiled Terminfo Entry
0  short     0433                Curses screen image
0  short     0434                Curses screen image
```

```
0  string    <ar>                System V Release 1 archive
0  string    !<arch>\n__.SYMDEF  Archive random library
0  string    !<arch>             Archive
0  string    ARF_BEGARF          PHIGS clear text archive
0  long      0x137A2950          Scalable OpenFont binary
0  long      0x137A2951          Encrypted scalable OpenFont binary
```

The use of a basic integer data type is intended to allow the implementation to choose a word size commonly used by applications on that architecture.

FUTURE DIRECTIONS

None.

SEE ALSO

ls

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

Options and an EXTENDED DESCRIPTION are added as specified in the IEEE P1003.2b draft standard.

IEEE PASC Interpretations 1003.2 #192 and #178 are applied.

17287 **NAME**

17288 find — find files

17289 **SYNOPSIS**

17290 `find [–H | –L] path ... [operand_expression ...]`

17291 **DESCRIPTION**

17292 The *find* utility shall recursively descend the directory hierarchy from each file specified by *path*,
17293 evaluating a Boolean expression composed of the primaries described in the OPERANDS section
17294 for each file encountered.

17295 The *find* utility shall be able to descend to arbitrary depths in a file hierarchy and shall not fail
17296 due to path length limitations (unless a *path* operand specified by the application exceeds
17297 {PATH_MAX} requirements).

17298 The *find* utility shall detect infinite loops; that is, entering a previously visited directory that is an
17299 ancestor of the last file encountered. When it detects an infinite loop, *find* shall write a
17300 diagnostic message to standard error and shall either recover its position in the hierarchy or
17301 terminate.

17302 **OPTIONS**

17303 The *find* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
17304 12.2, Utility Syntax Guidelines.

17305 The following options shall be supported by the implementation:

17306 **–H** Cause the file information and file type evaluated for each symbolic link
17307 encountered on the command line to be those of the file referenced by the link, and
17308 not the link itself. If the referenced file does not exist, the file information and type
17309 shall be for the link itself. File information for all symbolic links not on the
17310 command line shall be that of the link itself.

17311 **–L** Cause the file information and file type evaluated for each symbolic link to be
17312 those of the file referenced by the link, and not the link itself.

17313 Specifying more than one of the mutually-exclusive options **–H** and **–L** shall not be considered
17314 an error. The last option specified shall determine the behavior of the utility.

17315 **OPERANDS**

17316 The following operands shall be supported:

17317 The *path* operand is a pathname of a starting point in the directory hierarchy.

17318 The first argument that starts with a `'–'`, or is a `'!'` or a `'('`, and all subsequent arguments
17319 shall be interpreted as an *expression* made up of the following primaries and operators. In the
17320 descriptions, wherever *n* is used as a primary argument, it shall be interpreted as a decimal
17321 integer optionally preceded by a plus (`'+'`) or minus (`'–'`) sign, as follows:

17322 *+n* More than *n*.

17323 *n* Exactly *n*.

17324 *–n* Less than *n*.

17325 The following primaries shall be supported:

17326 **–name** *pattern*
17327 The primary shall evaluate as true if the basename of the filename being examined
17328 matches *pattern* using the pattern matching notation described in Section 2.13 (on
17329 page 2264).

17330 **–nouser** The primary shall evaluate as true if the file belongs to a user ID for which the
17331 *getpwuid*( ) function defined in the System Interfaces volume of
17332 IEEE Std 1003.1-200x (or equivalent) returns NULL.

17333 **–nogroup** The primary shall evaluate as true if the file belongs to a group ID for which the
17334 *getgrgid*( ) function defined in the System Interfaces volume of
17335 IEEE Std 1003.1-200x (or equivalent) returns NULL.

17336 **–xdev** The primary always shall evaluate as true; it shall cause *find* not to continue
17337 descending past directories that have a different device ID (*st_dev*, see the *stat*( )
17338 function defined in the System Interfaces volume of IEEE Std 1003.1-200x). If any
17339 **–xdev** primary is specified, it shall apply to the entire expression even if the **–xdev**
17340 primary would not normally be evaluated.

17341 **–prune** The primary always shall evaluate as true; it shall cause *find* not to descend the
17342 current pathname if it is a directory. If the **–depth** primary is specified, the **–prune**
17343 primary shall have no effect.

17344 **–perm [–]***mode*
17345 The *mode* argument is used to represent file mode bits. It shall be identical in
17346 format to the *symbolic_mode* operand described in *chmod* (on page 2438), and shall
17347 be interpreted as follows. To start, a template shall be assumed with all file mode
17348 bits cleared. An *op* symbol of `'+'` shall set the appropriate mode bits in the
17349 template; `'–'` shall clear the appropriate bits; `'='` shall set the appropriate mode
17350 bits, without regard to the contents of process' file mode creation mask. The *op*
17351 symbol of `'–'` cannot be the first character of *mode*; this avoids ambiguity with the
17352 optional leading hyphen. Since the initial mode is all bits off, there are not any
17353 symbolic modes that need to use `'–'` as the first character.

17354 If the hyphen is omitted, the primary shall evaluate as true when the file
17355 permission bits exactly match the value of the resulting template.

17356 Otherwise, if *mode* is prefixed by a hyphen, the primary shall evaluate as true if at
17357 least all the bits in the resulting template are set in the file permission bits.

17358 **–perm [–]***onum*
17359 If the hyphen is omitted, the primary shall evaluate as true when the file
17360 permission bits exactly match the value of the octal number *onum* and only the bits
17361 corresponding to the octal mask 07777 shall be compared. (See the description of
17362 the octal *mode* in *chmod* (on page 2438).) Otherwise, if *onum* is prefixed by a
17363 hyphen, the primary shall evaluate as true if at least all of the bits specified in *onum*
17364 that are also set in the octal mask 07777 are set.

17365 **–type** *c* The primary shall evaluate as true if the type of the file is *c*, where *c* is `'b'`, `'c'`,
17366 `'d'`, `'l'`, `'p'`, `'f'`, or `'s'` for block special file, character special file, directory,
17367 symbolic link, FIFO, regular file, or socket, respectively.

17368 **–links** *n* The primary shall evaluate as true if the file has *n* links.

17369 **–user** *uname* The primary shall evaluate as true if the file belongs to the user *uname*. If *uname* is
17370 a decimal integer and the *getpwnam*( ) (or equivalent) function does not return a
17371 valid user name, *uname* shall be interpreted as a user ID.

17372 **–group** *gname*
17373 The primary shall evaluate as true if the file belongs to the group *gname*. If *gname*
17374 is a decimal integer and the *getgrnam*( ) (or equivalent) function does not return a
17375 valid group name, *gname* shall be interpreted as a group ID.

17376 **–size** *n***[c]** The primary shall evaluate as true if the file size in bytes, divided by 512 and
17377 rounded up to the next integer, is *n*. If *n* is followed by the character 'c', the size
17378 shall be in bytes.

17379 **–atime** *n* The primary shall evaluate as true if the file access time subtracted from the
17380 initialization time, divided by 86 400 (with any remainder discarded), is *n*.

17381 **–ctime** *n* The primary shall evaluate as true if the time of last change of file status
17382 information subtracted from the initialization time, divided by 86 400 (with any
17383 remainder discarded), is *n*.

17384 **–mtime** *n* The primary shall evaluate as true if the file modification time subtracted from the
17385 initialization time, divided by 86 400 (with any remainder discarded), is *n*.

17386 **–exec** *utility_name* **[***argument* **...] ;**
17387 **–exec** *utility_name* **[***argument* **...] ;** { } +
17388 The end of the primary expression shall be punctuated by a semicolon or by a plus
17389 sign. Only a plus sign that follows an argument containing the two characters
17390 "{}" shall punctuate the end of the primary expression. Other uses of the plus
17391 sign shall not be treated as special.

17392 If the primary expression is punctuated by a semicolon, the utility *utility_name*
17393 shall be invoked once for each pathname and the primary shall evaluate as true if
17394 the utility returns a zero value as exit status. A *utility_name* or *argument* containing
17395 only the two characters "{}" shall be replaced by the current pathname.

17396 If the primary expression is punctuated by a plus sign, the primary shall always
17397 evaluate as true, and the pathnames for which the primary is evaluated shall be
17398 aggregated into sets. The utility *utility_name* shall be invoked once for each set of
17399 aggregated pathnames. Each invocation shall begin after the last pathname in the
17400 set is aggregated, and shall be completed before the *find* utility exits and before the
17401 first pathname in the next set (if any) is aggregated for this primary, but it is
17402 otherwise unspecified whether the invocation occurs before, during, or after the
17403 evaluations of other primaries. If any invocation returns a non-zero value as exit
17404 status, the *find* utility shall return a non-zero exit status. An argument containing
17405 only the two characters "{}" shall be replaced by the set of aggregated
17406 pathnames, with each pathname passed as a separate argument to the invoked
17407 utility in the same order that it was aggregated. The size of any set of two or more
17408 pathnames shall be limited such that execution of the utility does not cause the
17409 system's {ARG_MAX} limit to be exceeded. If more than one argument containing
17410 only the two characters "{}" is present, the behavior is unspecified.

17411 If a *utility_name* or *argument* string contains the two characters "{}", but not just
17412 the two characters "{}", it is implementation-defined whether *find* replaces those
17413 two characters or uses the string without change. The current directory for the
17414 invocation of *utility_name* shall be the same as the current directory when the *find*
17415 utility was started. If the *utility_name* names any of the special built-in utilities (see
17416 Section 2.14 (on page 2266)), the results are undefined.

17417 **–ok** *utility_name* **[***argument* **...] ;**
17418 The **–ok** primary shall be equivalent to **–exec**, except that the use of a plus sign to
17419 punctuate the end of the primary expression need not be supported, and *find* shall
17420 request affirmation of the invocation of *utility_name* using the current file as an
17421 argument by writing to standard error as described in the STDERR section. If the
17422 response on standard input is affirmative, the utility shall be invoked. Otherwise,
17423 the command shall not be invoked and the value of the **–ok** operand shall be false.

17424 **–print** The primary always shall evaluate as true; it shall cause the current pathname to
17425 be written to standard output.

17426 **–newer** *file* The primary shall evaluate as true if the modification time of the current file is
17427 more recent than the modification time of the file named by the pathname *file*.

17428 **–depth** The primary shall always evaluate as true; it shall cause descent of the directory
17429 hierarchy to be done so that all entries in a directory are acted on before the
17430 directory itself. If a **–depth** primary is not specified, all entries in a directory shall
17431 be acted on after the directory itself. If any **–depth** primary is specified, it shall
17432 apply to the entire expression even if the **–depth** primary would not normally be
17433 evaluated.

17434 The primaries can be combined using the following operators (in order of decreasing
17435 precedence):

17436 (*expression*) True if *expression* is true.

17437 **!** *expression* Negation of a primary; the unary NOT operator.

17438 *expression* **[–a]** *expression*
17439 Conjunction of primaries; the AND operator is implied by the juxtaposition of two
17440 primaries or made explicit by the optional **–a** operator. The second expression
17441 shall not be evaluated if the first expression is false.

17442 *expression* **–o** *expression*
17443 Alternation of primaries; the OR operator. The second expression shall not be
17444 evaluated if the first expression is true.

17445 If no *expression* is present, **–print** shall be used as the expression. Otherwise, if the given
17446 expression does not contain any of the primaries **–exec**, **–ok**, or **–print**, the given expression shall
17447 be effectively replaced by:

17448 `( given_expression ) –print`

17449 The **–user**, **–group**, and **–newer** primaries each shall evaluate their respective arguments only
17450 once.

17451 **STDIN**

17452 If the **–ok** primary is used, the response shall be read from the standard input. An entire line
17453 shall be read as the response. Otherwise, the standard input shall not be used.

17454 **INPUT FILES**

17455 None.

17456 **ENVIRONMENT VARIABLES**

17457 The following environment variables shall affect the execution of *find*:

17458 *LANG* Provide a default value for the internationalization variables that are unset or null.
17459 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
17460 Internationalization Variables for the precedence of internationalization variables
17461 used to determine the values of locale categories.)

17462 *LC_ALL* If set to a non-empty string value, override the values of all the other
17463 internationalization variables.

17464 *LC_COLLATE*
17465 Determine the locale for the behavior of ranges, equivalence classes and multi-
17466 character collating elements used in the pattern matching notation for the **–n**
17467 option and in the extended regular expression defined for the **yesexpr** locale

keyword in the *LC_MESSAGES* category.

LC_CTYPE This variable determines the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments), the behavior of character classes within the pattern matching notation used for the **–n** option, and the behavior of character classes within regular expressions used in the extended regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES* category.

LC_MESSAGES
Determine the locale for the processing of affirmative responses that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

PATH Determine the location of the *utility_name* for the **–exec** and **–ok** primaries, as described in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The **–print** primary shall cause the current pathnames to be written to standard output. The format shall be:

```
"%s\n", <path>
```

STDERR

The **–ok** primary shall write a prompt to standard error containing at least the *utility_name* to be invoked and the current pathname. In the POSIX locale, the last non-<blank> in the prompt shall be `'?'`. The exact format used is unspecified.

Otherwise, the standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All *path* operands were traversed successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

When used in operands, pattern matching notation, semicolons, opening parentheses, and closing parentheses are special to the shell and must be quoted (see Section 2.2 (on page 2232)).

The bit that is traditionally used for sticky (historically 01000) is specified in the –**perm** primary using the octal number argument form. Since this bit is not defined by this volume of IEEE Std 1003.1-200x, applications must not assume that it actually refers to the traditional sticky bit.

EXAMPLES

1. The following commands are equivalent:

   ```
   find .
   find . –print
   ```

   They both write out the entire directory hierarchy from the current directory.

2. The following command:

   ```
   find / \( –name tmp –o –name '*.xx' \) –atime +7 –exec rm {} \;
   ```

   removes all files named **tmp** or ending in **.xx** that have not been accessed for seven or more 24-hour periods.

3. The following command:

   ```
   find . –perm –o+w,+s
   ```

   prints (–**print** is assumed) the names of all files in or below the current directory, with all of the file permission bits S_ISUID, S_ISGID, and S_IWOTH set.

4. The following command:

   ```
   find . –name SCCS –prune –o –print
   ```

   recursively prints pathnames of all files in the current directory and below, but skips directories named SCCS and files in them.

5. The following command:

   ```
   find . –print –name SCCS –prune
   ```

   behaves as in the previous example, but prints the names of the SCCS directories.

6. The following command is roughly equivalent to the –**nt** extension to *test*:

   ```
   if [ –n "$(find file1 –prune –newer file2)" ]; then
       printf %s\\n "file1 is newer than file2"
   fi
   ```

7. The descriptions of –**atime**, –**ctime**, and –**mtime** use the terminology *n* ''86 400 second periods (days)''. For example, a file accessed at 23:59 is selected by:

   ```
   find . –atime –1 –print
   ```

   at 00:01 the next day (less than 24 hours later, not more than one day ago); the midnight boundary between days has no effect on the 24-hour calculation.

RATIONALE

The –**a** operator was retained as an optional operator for compatibility with historical shell scripts, even though it is redundant with expression concatenation.

The descriptions of the '–' modifier on the *mode* and *onum* arguments to the –**perm** primary agree with historical practice on BSD and System V implementations. System V and BSD documentation both describe it in terms of checking additional bits; in fact, it uses the same bits, but checks for having at least all of the matching bits set instead of having exactly the matching bits set.

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified because:

- Implementations may desire more descriptive prompts than those used on historical implementations.
- Since the historical prompt strings do not terminate with <newline>s, there is no portable way for another program to interact with the prompts of this utility via pipes.

Therefore, an application using this prompting option relies on the system to provide the most suitable dialog directly with the user, based on the general guidelines specified.

The –**name** *file* operand was changed to use the shell pattern matching notation so that *find* is consistent with other utilities using pattern matching.

The –**size** operand refers to the size of a file, rather than the number of blocks it may occupy in the file system. The intent is that the *st_size* field defined in the System Interfaces volume of IEEE Std 1003.1-200x should be used, not the *st_blocks* found in historical implementations. There are at least two reasons for this:

1. In both System V and BSD, *find* only uses *st_size* in size calculations for the operands specified by this volume of IEEE Std 1003.1-200x. (BSD uses *st_blocks* only when processing the –**ls** primary.)
2. Users usually think of file size in terms of bytes, which is also the unit used by the *ls* utility for the output from the –**l** option. (In both System V and BSD, *ls* uses *st_size* for the –**l** option size field and uses *st_blocks* for the *ls* –**s** calculations. This volume of IEEE Std 1003.1-200x does not specify *ls* –**s**.)

The descriptions of –**atime**, –**ctime**, and –**mtime** were changed from the SVID description of *n* ''days'' to ''24-hour periods''. The description is also different in terms of the exact timeframe for the *n* case (*versus* the +*n* or –*n*), but it matches all known historical implementations. It refers to one 86 400 second period in the past, not any time from the beginning of that period to the current time. For example, –**atime** 3 is true if the file was accessed any time in the period from 72 hours to 48 hours ago.

Historical implementations do not modify "{}" when it appears as a substring of an –**exec** or –**ok** *utility_name* or argument string. There have been numerous user requests for this extension, so this volume of IEEE Std 1003.1-200x allows the desired behavior. At least one recent implementation does support this feature, but encountered several problems in managing memory allocation and dealing with multiple occurrences of "{}" in a string while it was being developed, so it is not yet required behavior.

Assuming the presence of –**print** was added to correct a historical pitfall that plagues novice users, it is entirely upward-compatible from the historical System V *find* utility. In its simplest form (*find directory*), it could be confused with the historical BSD fast *find*. The BSD developers agreed that adding –**print** as a default expression was the correct decision and have added the fast *find* functionality within a new utility called *locate*.

Historically, the –**L** option was implemented using the primary –**follow**. The –**H** and –**L** options were added for two reasons. First, they offer a finer granularity of control and consistency with other programs that walk file hierarchies. Second, the –**follow** primary always evaluated to true.

As they were historically really global variables that took effect before the traversal began, some valid expressions had unexpected results. An example is the expression **–print –o –follow**. Because **–print** always evaluates to true, the standard order of evaluation implies that **–follow** would never be evaluated. This was never the case. Historical practice for the **–follow** primary, however, is not consistent. Some implementations always follow symbolic links on the command line whether **–follow** is specified or not. Others follow symbolic links on the command line only if **–follow** is specified. Both behaviors are provided by the **–H** and **–L** options, but scripts using the current **–follow** primary would be broken if the **–follow** option is specified to work either way.

Since the **–L** option resolves all symbolic links and the **–type** *l* primary is true for symbolic links that still exist after symbolic links have been resolved, the command:

```
find –L . –type l
```

prints a list of symbolic links reachable from the current directory that do not resolve to accessible files.

A feature of SVR4's *find* utility was the **–exec** primary's **+** terminator. This allowed filenames containing special characters (especially <newline>s) to be grouped together without the problems that occur if such filenames are piped to *xargs*. Other implementations have added other ways to get around this problem, notably a **–print0** primary that wrote filenames with a null byte terminator. This was considered here, but not adopted. Using a null terminator meant that any utility that was going to process *find*'s **–print0** output had to add a new option to parse the null terminators it would now be reading.

The "`–exec ... {} +`" syntax adopted was a result of IEEE PASC Interpretation 1003.2 #210. It should be noted that this is an incompatible change to the ISO/IEC 9899:1999 standard. For example, the following command prints all files with a `'–'` after their name if they are regular files, and a `'+'` otherwise:

```
find / –type f –exec echo {} – ';' –o –exec echo {} + ';'
```

The change invalidates usage like this. Even though the previous standard stated that this usage would work, in practice many did not support it and the standard developers felt it better to now state that this was not allowable.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod, *pax*, *sh*, *test*, the System Interfaces volume of IEEE Std 1003.1-200x, *stat*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

FUTURE DIRECTIONS section added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **–perm [–]***onum* primary is supported.

The *find* utility is aligned with the IEEE P1003.2b draft standard, to include processing of symbolic links and changes to the description of the **atime**, **ctime**, and **mtime** operands.

IEEE PASC Interpretation 1003.2 #210 is applied, extending the **–exec** operand.

17633 **NAME**

17634 fold — filter for folding lines

17635 **SYNOPSIS**

17636 `fold [–bs][–w width][file...]`

17637 **DESCRIPTION**

17638 The *fold* utility is a filter that shall fold lines from its input files, breaking the lines to have a
17639 maximum of *width* column positions (or bytes, if the **–b** option is specified). Lines shall be
17640 broken by the insertion of a <newline> such that each output line (referred to later in this section
17641 as a *segment*) is the maximum width possible that does not exceed the specified number of
17642 column positions (or bytes). A line shall not be broken in the middle of a character. The behavior
17643 is undefined if *width* is less than the number of columns any single character in the input would
17644 occupy.

17645 If the <carriage-return>s, <backspace>s, or <tab>s are encountered in the input, and the **–b**
17646 option is not specified, they shall be treated specially:

17647 <backspace> The current count of line width shall be decremented by one, although the count
17648 never shall become negative. The *fold* utility shall not insert a <newline>
17649 immediately before or after any <backspace>.

17650 <carriage-return>
17651 The current count of line width shall be set to zero. The *fold* utility shall not insert a
17652 <newline> immediately before or after any <carriage-return>.

17653 <tab> Each <tab> encountered shall advance the column position pointer to the next tab
17654 stop. Tab stops shall be at each column position *n* such that *n* modulo 8 equals 1.

17655 **OPTIONS**

17656 The *fold* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
17657 12.2, Utility Syntax Guidelines.

17658 The following options shall be supported:

17659 **–b** Count *width* in bytes rather than column positions.

17660 **–s** If a segment of a line contains a <blank> within the first *width* column positions (or
17661 bytes), break the line after the last such <blank> meeting the width constraints. If
17662 there is no <blank> meeting the requirements, the **–s** option shall have no effect for
17663 that output segment of the input line.

17664 **–w** *width* Specify the maximum line length, in column positions (or bytes if **–b** is specified).
17665 The results are unspecified if *width* is not a positive decimal number. The default
17666 value shall be 80.

17667 **OPERANDS**

17668 The following operand shall be supported:

17669 *file* A pathname of a text file to be folded. If no *file* operands are specified, the standard
17670 input shall be used.

17671 **STDIN**

17672 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES
17673 section.

17674 **INPUT FILES**

17675 If the **–b** option is specified, the input files shall be text files except that the lines are not limited
17676 to {LINE_MAX} bytes in length. If the **–b** option is not specified, the input files shall be text files.

17677 **ENVIRONMENT VARIABLES**

17678 The following environment variables shall affect the execution of *fold*:

17679 *LANG* Provide a default value for the internationalization variables that are unset or null.
17680 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
17681 Internationalization Variables for the precedence of internationalization variables
17682 used to determine the values of locale categories.)

17683 *LC_ALL* If set to a non-empty string value, override the values of all the other
17684 internationalization variables.

17685 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
17686 characters (for example, single-byte as opposed to multi-byte characters in
17687 arguments and input files), and for the determination of the width in column
17688 positions each character would occupy on a constant-width font output device.

17689 *LC_MESSAGES*
17690 Determine the locale that should be used to affect the format and contents of
17691 diagnostic messages written to standard error.

17692 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

17693 **ASYNCHRONOUS EVENTS**

17694 Default.

17695 **STDOUT**

17696 The standard output shall be a file containing a sequence of characters whose order shall be
17697 preserved from the input files, possibly with inserted <newline>s.

17698 **STDERR**

17699 The standard error shall be used only for diagnostic messages.

17700 **OUTPUT FILES**

17701 None.

17702 **EXTENDED DESCRIPTION**

17703 None.

17704 **EXIT STATUS**

17705 The following exit values shall be returned:

17706 0 All input files were processed successfully.

17707 >0 An error occurred.

17708 **CONSEQUENCES OF ERRORS**

17709 Default.

APPLICATION USAGE

The *cut* and *fold* utilities can be used to create text files out of files with arbitrary line lengths. The *cut* utility should be used when the number of lines (or records) needs to remain constant. The *fold* utility should be used when the contents of long lines need to be kept contiguous.

The *fold* utility is frequently used to send text files to printers that truncate, rather than fold, lines wider than the printer is able to print (usually 80 or 132 column positions).

EXAMPLES

An example invocation that submits a file of possibly long lines to the printer (under the assumption that the user knows the line width of the printer to be assigned by *lp*):

```
fold –w 132 bigfile | lp
```

RATIONALE

Although terminal input in canonical processing mode requires the erase character (frequently set to <backspace>) to erase the previous character (not byte or column position), terminal output is not buffered and is extremely difficult, if not impossible, to parse correctly; the interpretation depends entirely on the physical device that actually displays/prints/stores the output. In all known internationalized implementations, the utilities producing output for mixed column-width output assume that a <backspace> backs up one column position and outputs enough <backspace>s to return to the start of the character when <backspace> is used to provide local line motions to support underlining and emboldening operations. Since *fold* without the **–b** option is dealing with these same constraints, <backspace> is always treated as backing up one column position rather than backing up one character.

Historical versions of the *fold* utility assumed 1 byte was one character and occupied one column position when written out. This is no longer always true. Since the most common usage of *fold* is believed to be folding long lines for output to limited-length output devices, this capability was preserved as the default case. The **–b** option was added so that applications could *fold* files with arbitrary length lines into text files that could then be processed by the standard utilities. Note that although the width for the **–b** option is in bytes, a line is never split in the middle of a character. (It is unspecified what happens if a width is specified that is too small to hold a single character found in the input followed by a <newline>.)

The tab stops are hardcoded to be every eighth column to meet historical practice. No new method of specifying other tab stops was invented.

FUTURE DIRECTIONS

None.

SEE ALSO

cut

CHANGE HISTORY

First released in Issue 4.

Issue 6

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

NAME

fort77 — FORTRAN compiler (**FORTRAN**)

SYNOPSIS

FD
```
fort77 [–c][–g][–L directory]... [–O optlevel][–o outfile][–s][–w]
    operand...
```

DESCRIPTION

The *fort77* utility is the interface to the FORTRAN compilation system; it shall accept the full FORTRAN-77 language defined by the ANSI X3.9-1978 standard. The system conceptually consists of a compiler and link editor. The files referenced by *operand*s are compiled and linked to produce an executable file. It is unspecified whether the linking occurs entirely within the operation of *fort77*; some implementations may produce objects that are not fully resolved until the file is executed.

If the **–c** option is present, for all pathname operands of the form *file***.f**, the files:

```
$(basename pathname.f).o
```

shall be created or overwritten as the result of successful compilation. If the **–c** option is not specified, it is unspecified whether such **.o** files are created or deleted for the *file***.f** operands.

If there are no options that prevent link editing (such as **–c**) and all operands compile and link without error, the resulting executable file shall be written into the file named by the **–o** option (if present) or to the file **a.out**. The executable file shall be created as specified in the System Interfaces volume of IEEE Std 1003.1-200x, except that the file permissions shall be set to:

S_IRWXO | S_IRWXG | S_IRWXU

and that the bits specified by the *umask* of the process shall be cleared.

OPTIONS

The *fort77* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines, except that:

- The **–l** *library* operands have the format of options, but their position within a list of operands affects the order in which libraries are searched.
- The order of specifying the multiple **–L** options is significant.
- Conforming applications shall specify each option separately; that is, grouping option letters (for example, **–cg**) need not be recognized by all implementations.

The following options shall be supported:

–c Suppress the link-edit phase of the compilation, and do not remove any object files that are produced.

–g Produce symbolic information in the object or executable files; the nature of this information is unspecified, and may be modified by implementation-defined interactions with other options.

–s Produce object or executable files, or both, from which symbolic and other information not required for proper execution using the *exec* family of functions defined in the System Interfaces volume of IEEE Std 1003.1-200x has been removed (stripped). If both **–g** and **–s** options are present, the action taken is unspecified.

–o *outfile* Use the pathname *outfile*, instead of the default **a.out**, for the executable file produced. If the **–o** option is present with **–c**, the result is unspecified.

17792 **–L** *directory* Change the algorithm of searching for the libraries named in **–l** operands to look in
17793 the directory named by the *directory* pathname before looking in the usual places.
17794 Directories named in **–L** options shall be searched in the specified order. At least
17795 ten instances of this option shall be supported in a single *fort77* command
17796 invocation. If a directory specified by a **–L** option contains a file named **libf.a**, the
17797 results are unspecified.

17798 **–O** *optlevel* Specify the level of code optimization. If the *optlevel* option-argument is the digit
17799 `'0'`, all special code optimizations shall be disabled. If it is the digit `'1'`, the
17800 nature of the optimization is unspecified. If the **–O** option is omitted, the nature of
17801 the system's default optimization is unspecified. It is unspecified whether code
17802 generated in the presence of the **–O** 0 option is the same as that generated when
17803 **–O** is omitted. Other *optlevel* values may be supported.

17804 **–w** Suppress warnings.

17805 Multiple instances of **–L** options can be specified.

17806 **OPERANDS**

17807 An *operand* is either in the form of a pathname or the form **–l** *library*. At least one operand of the
17808 pathname form shall be specified. The following operands shall be supported:

17809 *file*.**f** The pathname of a FORTRAN source file to be compiled and optionally passed to
17810 the link editor. The filename operand shall be of this form if the **–c** option is used.

17811 *file*.**a** A library of object files typically produced by *ar*, and passed directly to the link
17812 editor. Implementations may recognize implementation-defined suffixes other
17813 than **.a** as denoting object file libraries.

17814 *file*.**o** An object file produced by *fort77* **–c** and passed directly to the link editor.
17815 Implementations may recognize implementation-defined suffixes other than **.o** as
17816 denoting object files.

17817 The processing of other files is implementation-defined.

17818 **–l** *library* (The letter ell.) Search the library named:

17819 `lib`*`library`*`.a`

17820 A library is searched when its name is encountered, so the placement of a **–l**
17821 operand is significant. Several standard libraries can be specified in this manner, as
17822 described in the EXTENDED DESCRIPTION section. Implementations may
17823 recognize implementation-defined suffixes other than **.a** as denoting libraries.

17824 **STDIN**

17825 Not used.

17826 **INPUT FILES**

17827 The input file shall be one of the following: a text file containing FORTRAN source code; an
17828 object file in the format produced by *fort77* **–c**; or a library of object files, in the format produced
17829 by archiving zero or more object files, using *ar*. Implementations may supply additional utilities
17830 that produce files in these formats. Additional input files are implementation-defined.

17831 A <tab> encountered within the first six characters on a line of source code shall cause the
17832 compiler to interpret the following character as if it were the seventh character on the line (that
17833 is, in column 7).

17834 **ENVIRONMENT VARIABLES**

17835 The following environment variables shall affect the execution of *fort77*:

17836 *LANG* Provide a default value for the internationalization variables that are unset or null.
17837 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
17838 Internationalization Variables for the precedence of internationalization variables
17839 used to determine the values of locale categories.)

17840 *LC_ALL* If set to a non-empty string value, override the values of all the other
17841 internationalization variables.

17842 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
17843 characters (for example, single-byte as opposed to multi-byte characters in
17844 arguments and input files).

17845 *LC_MESSAGES*
17846 Determine the locale that should be used to affect the format and contents of
17847 diagnostic messages written to standard error.

17848 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

17849 *TMPDIR* Determine the pathname that should override the default directory for temporary
17850 files, if any.

17851 **ASYNCHRONOUS EVENTS**

17852 Default.

17853 **STDOUT**

17854 Not used.

17855 **STDERR**

17856 The standard error shall be used only for diagnostic messages. If more than one *file* operand |
17857 ending in **.f** (or possibly other unspecified suffixes) is given, for each such file:

17858 `"%s:\n", <file>`

17859 may be written to allow identification of the diagnostic message with the appropriate input file.

17860 This utility may produce warning messages about certain conditions that do not warrant
17861 returning an error (non-zero) exit value.

17862 **OUTPUT FILES**

17863 Object files, listing files and executable files shall be produced in unspecified formats.

17864 **EXTENDED DESCRIPTION**

17865 **Standard Libraries**

17866 The *fort77* utility shall recognize the following **–l** operand for the standard library:

17867 **–l f** This library contains all functions referenced in the ANSI X3.9-1978 standard. This
17868 operand shall not be required to be present to cause a search of this library.

17869 In the absence of options that inhibit invocation of the link editor, such as **–c**, the *fort77* utility
17870 shall cause the equivalent of a **–l f** operand to be passed to the link editor as the last **–l** operand,
17871 causing it to be searched after all other object files and libraries are loaded.

17872 It is unspecified whether the library **libf.a** exists as a regular file. The implementation may
17873 accept as **–l** operands names of objects that do not exist as regular files.

External Symbols

The FORTRAN compiler and link editor shall support the significance of external symbols up to a length of at least 31 bytes; case folding is permitted. The action taken upon encountering symbols exceeding the implementation-defined maximum symbol length is unspecified.

The compiler and link editor shall support a minimum of 511 external symbols per source or object file, and a minimum of 4 095 external symbols total. A diagnostic message is written to standard output if the implementation-defined limit is exceeded; other actions are unspecified.

EXIT STATUS

The following exit values shall be returned:

0 Successful compilation or link edit.

>0 An error occurred.

CONSEQUENCES OF ERRORS

When *fort77* encounters a compilation error, it shall write a diagnostic to standard error and continue to compile other source code operands. It shall return a non-zero exit status, but it is implementation-defined whether an object module is created. If the link edit is unsuccessful, a diagnostic message shall be written to standard error, and *fort77* shall exit with a non-zero status.

APPLICATION USAGE

None.

EXAMPLES

The following usage example compiles **xyz.f** and creates the executable file **foo**:

```
fort77 –o foo xyz.f
```

The following example compiles **xyz.f** and creates the object file **xyz.o**:

```
fort77 –c xyz.f
```

The following example compiles **xyz.f** and creates the executable file **a.out**:

```
fort77 xyz.f
```

The following example compiles **xyz.f**, links it with **b.o**, and creates the executable **a.out**:

```
fort77 xyz.f b.o
```

RATIONALE

The name of this utility was chosen as *fort77* to parallel the renaming of the C compiler. The name *f77* was not chosen to avoid problems with historical implementations. The ANSI X3.9-1978 standard was selected as a normative reference because the ISO / IEC version of FORTRAN-77 has been superseded by the ISO / IEC 1539: 1990 standard (Fortran-90).

The file inclusion and symbol definition **#define** mechanisms used by the *c99* utility were not included in this volume of IEEE Std 1003.1-200x—even though they are commonly implemented—since there is no requirement that the FORTRAN compiler use the C preprocessor.

The **–onetrip** option was not included in this volume of IEEE Std 1003.1-200x, even though many historical compilers support it, because it is derived from FORTRAN-66; it is an anachronism that should not be perpetuated.

Some implementations produce compilation listings. This aspect of FORTRAN has been left unspecified because there was controversy concerning the various methods proposed for implementing it: a **–V** option overlapped with historical vendor practice and a naming

convention of creating files with **.l** suffixes collided with historical *lex* file naming practice.

There is no **–l** option in this version of this volume of IEEE Std 1003.1-200x to specify a directory for file inclusion. An INCLUDE directive has been a part of the Fortran-90 discussions, but an interface supporting that standard is not in the current scope.

It is noted that many FORTRAN compilers produce an object module even when compilation errors occur; during a subsequent compilation, the compiler may patch the object module rather than recompiling all the code. Consequently, it is left to the implementor whether or not an object file is created.

A reference to MIL-STD-1753 was removed from an early proposal in response to a request from the POSIX FORTRAN-binding standard developers. It was not the intention of the standard developers to require certification of the FORTRAN compiler, and IEEE Std 1003.9-1992 does not specify the military standard or any special preprocessing requirements. Furthermore, use of that document would have been inappropriate for an international standard.

The specification of optimization has been subject to changes through early proposals. At one time, **–O** and **–N** were Booleans: optimize and do not optimize (with an unspecified default). Some historical practice lead this to be changed to:

–O 0 No optimization.

–O 1 Some level of optimization.

–O *n* Other, unspecified levels of optimization.

It is not always clear whether ‘‘good code generation’’ is the same thing as optimization. Simple optimizations of local actions do not usually affect the semantics of a program. The **–O** 0 option has been included to accommodate the very particular nature of scientific calculations in a highly optimized environment; compilers make errors. Some degree of optimization is expected, even if it is not documented here, and the ability to shut it off completely could be important when porting an application. An implementation may treat **–O** 0 as ‘‘do less than normal’’ if it wishes, but this is only meaningful if any of the operations it performs can affect the semantics of a program. It is highly dependent on the implementation whether doing less than normal is logical. It is not the intent of the **–O** 0 option to ask for inefficient code generation, but rather to assure that any semantically visible optimization is suppressed.

The specification of standard library access is consistent with the C compiler specification. Implementations are not required to have **/usr/lib/libf.a**, as many historical implementations do, but if not they are required to recognize **f** as a token.

External symbol size limits are in normative text; conforming applications need to know these limits. However, the minimum maximum symbol length should be taken as a constraint on a conforming application, not on an implementation, and consequently the action taken for a symbol exceeding the limit is unspecified. The minimum size for the external symbol table was added for similar reasons.

The CONSEQUENCES OF ERRORS section clearly specifies the behavior of the compiler when compilation or link-edit errors occur. The behavior of several historical implementations was examined, and the choice was made to be silent on the status of the executable, or **a.out**, file in the face of compiler or linker errors. If a linker writes the executable file, then links it on disk with *lseek*( )s and *write*( )s, the partially linked executable file can be left on disk and its execute bits turned off if the link edit fails. However, if the linker links the image in memory before writing the file to disk, it need not touch the executable file (if it already exists) because the link edit fails. Since both approaches are historical practice, a conforming application shall rely on the exit status of *fort77*, rather than on the existence or mode of the executable file.

The **–g** and **–s** options are not specified as mutually-exclusive. Historically these two options have been mutually-exclusive, but because both are so loosely specified, it seemed appropriate to leave their interaction unspecified.

The requirement that conforming applications specify compiler options separately is to reserve the multi-character option name space for vendor-specific compiler options, which are known to exist in many historical implementations. Implementations are not required to recognize, for example, **–gc** as if it were **–g –c**; nor are they forbidden from doing so. The SYNOPSIS shows all of the options separately to highlight this requirement on applications.

Echoing filenames to standard error is considered a diagnostic message because it would otherwise be difficult to associate an error message with the erring file. They are described with ''may'' to allow implementations to use other methods of identifying files and to parallel the description in *c99*.

FUTURE DIRECTIONS

A compilation system based on the ISO/IEC 1539:1990 standard (Fortran-90) may be considered for a future issue; it may have a different utility name from *fort77*.

SEE ALSO

ar, *asa*, *c99*, *umask*

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is now marked as part of the FORTRAN Development Utilities option.

The normative text is reworded to avoid use of the term ''must'' for application requirements.

17985 **NAME**

17986 fuser — list process IDs of all processes that have one or more files open

17987 **SYNOPSIS**

17988 XSI `fuser [ –cfu ] file ...`

17989

17990 **DESCRIPTION**

17991 The *fuser* utility shall write to standard output the process IDs of processes running on the local
17992 system that have one or more named files open. For block special devices, all processes using
17993 any file on that device are listed.

17994 The *fuser* utility shall write to standard error additional information about the named files
17995 indicating how the file is being used.

17996 Any output for processes running on remote systems that have a named file open is unspecified.

17997 A user may need appropriate privilege to invoke the *fuser* utility.

17998 **OPTIONS**

17999 The *fuser* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
18000 12.2, Utility Syntax Guidelines.

18001 The following options shall be supported:

18002 **–c** The file is treated as a mount point and the utility shall report on any files open in
18003 the file system.

18004 **–f** The report shall be only for the named files.

18005 **–u** The user name, in parentheses, associated with each process ID written to standard
18006 output shall be written to standard error.

18007 **OPERANDS**

18008 The following operand shall be supported:

18009 *file* A pathname on which the file or file system is to be reported.

18010 **STDIN**

18011 Not used.

18012 **INPUT FILES**

18013 The user database.

18014 **ENVIRONMENT VARIABLES**

18015 The following environment variables shall affect the execution of *fuser*:

18016 *LANG* Provide a default value for the internationalization variables that are unset or null.
18017 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
18018 Internationalization Variables for the precedence of internationalization variables
18019 used to determine the values of locale categories.)

18020 *LC_ALL* If set to a non-empty string value, override the values of all the other
18021 internationalization variables.

18022 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
18023 characters (for example, single-byte as opposed to multi-byte characters in
18024 arguments).

18025 *LC_MESSAGES*
18026 Determine the locale that should be used to affect the format and contents of
18027 diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The *fuser* utility shall write the process ID for each process using each file given as an operand to standard output in the following format:

```
"%d", <process_id>
```

STDERR

The *fuser* utility shall write diagnostic messages to standard error.

The *fuser* utility also shall write the following to standard error:

- The pathname of each named file is written followed immediately by a colon.
- For each process ID written to standard output, the character `'c'` shall be written to standard error if the process is using the file as its current directory and the character `'r'` shall be written to standard error if the process is using the file as its root directory. Implementations may write other alphabetic characters to indicate other uses of files.
- When the **–u** option is specified, characters indicating the use of the file shall be followed immediately by the user name, in parentheses, corresponding to the process' real user ID. If the user name cannot be resolved from the process' real user ID, the process' real user ID shall be written instead of the user name.

When standard output and standard error are directed to the same file, the output shall be interleaved so that the filename appears at the start of each line, followed by the process ID and characters indicating the use of the file. Then, if the **–u** option is specified, the user name or user ID for each process using that file shall be written.

A <newline> shall be written to standard error after the last output described above for each *file* operand.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

The command:

```
fuser –fu .
```

writes to standard output the process IDs of processes that are using the current directory and writes to standard error an indication of how those processes are using the directory and the user names associated with the processes that are using the current directory.

RATIONALE

The definition of the *fuser* utility follows existing practice.

FUTURE DIRECTIONS

None.

SEE ALSO

None.

CHANGE HISTORY

First released in Issue 5.

18079 **NAME**

18080 gencat — generate a formatted message catalog

18081 **SYNOPSIS**

18082 XSI
```
gencat catfile msgfile...
```
18083

18084 **DESCRIPTION**

18085 The *gencat* utility shall merge the message text source files *msgfile* into a formatted message
18086 catalog *catfile*. The file *catfile* shall be created if it does not already exist. If *catfile* does exist, its
18087 messages shall be included in the new *catfile*. If set and message numbers collide, the new
18088 message text defined in *msgfile* shall replace the old message text currently contained in *catfile*.

18089 **OPTIONS**

18090 None.

18091 **OPERANDS**

18092 The following operands shall be supported:

18093 *catfile* A pathname of the formatted message catalog. If '−' is specified, standard output
18094 shall be used. The format of the message catalog produced is unspecified.

18095 *msgfile* A pathname of a message text source file. If '−' is specified for an instance of
18096 *msgfile*, standard input shall be used. The format of message text source files is
18097 defined in the EXTENDED DESCRIPTION section.

18098 **STDIN**

18099 The standard input shall not be used unless a *msgfile* operand is specified as '−'.

18100 **INPUT FILES**

18101 The input files shall be text files.

18102 **ENVIRONMENT VARIABLES**

18103 The following environment variables shall affect the execution of *gencat*:

18104 *LANG* Provide a default value for the internationalization variables that are unset or null.
18105 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
18106 Internationalization Variables for the precedence of internationalization variables
18107 used to determine the values of locale categories.)

18108 *LC_ALL* If set to a non-empty string value, override the values of all the other
18109 internationalization variables.

18110 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
18111 characters (for example, single-byte as opposed to multi-byte characters in
18112 arguments and input files).

18113 *LC_MESSAGES*
18114 Determine the locale that should be used to affect the format and contents of
18115 diagnostic messages written to standard error.

18116 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

18117 **ASYNCHRONOUS EVENTS**

18118 Default.

18119 **STDOUT**

18120 The standard output shall not be used unless the *catfile* operand is specified as '−'.

18121 **STDERR**

18122 The standard error shall be used only for diagnostic messages.

18123 **OUTPUT FILES**

18124 None.

18125 **EXTENDED DESCRIPTION**

18126 The content of a message text file shall be in the format defined as follows. Note that the fields of
18127 a message text source line are separated by a single <blank>. Any other <blank>s are considered
18128 as being part of the subsequent field.

18129 **$set** *n comment*

18130 This line specifies the set identifier of the following messages until the next **$set** or
18131 end-of-file appears. The *n* denotes the set identifier, which is defined as a number
18132 in the range [1, {NL_SETMAX}] (see the **<limits.h>** header defined in the System
18133 Interfaces volume of IEEE Std 1003.1-200x). The application shall ensure that set
18134 identifiers are presented in ascending order within a single source file, but need
18135 not be contiguous. Any string following the set identifier shall be treated as a
18136 comment. If no **$set** directive is specified in a message text source file, all messages
18137 shall be located in an implementation-defined default message set NL_SETD (see
18138 the **<nl_types.h>** header defined in the System Interfaces volume of
18139 IEEE Std 1003.1-200x).

18140 **$delset** *n comment*

18141 This line deletes message set *n* from an existing message catalog. The *n* denotes the
18142 set number [1, {NL_SETMAX}]. Any string following the set number shall be
18143 treated as a comment.

18144 **$** *comment* A line beginning with '$' followed by a <blank> shall be treated as a comment.

18145 *m message-text*

18146 The *m* denotes the message identifier, which is defined as a number in the range [1,
18147 {NL_MSGMAX}] (see the **<limits.h>** header defined in the System Interfaces
18148 volume of IEEE Std 1003.1-200x). The *message-text* shall be stored in the message
18149 catalog with the set identifier specified by the last **$set** directive, and with message
18150 identifier *m*. If the *message-text* is empty, and a <blank> field separator is present,
18151 an empty string shall be stored in the message catalog. If a message source line has
18152 a message number, but neither a field separator nor *message-text*, the existing
18153 message with that number (if any) shall be deleted from the catalog. The
18154 application shall ensure that message identifiers are in ascending order within a
18155 single set, but need not be contiguous. The application shall ensure that the length
18156 of *message-text* is in the range [0, {NL_TEXTMAX}] (see the **<limits.h>** header
18157 defined in the System Interfaces volume of IEEE Std 1003.1-200x).

18158 **$quote** *n* This line specifies an optional quote character *c*, which can be used to surround
18159 *message-text* so that trailing spaces or null (empty) messages are visible in a
18160 message source line. By default, or if an empty **$quote** directive is supplied, no
18161 quoting of *message-text* shall be recognized.

18162 Empty lines in a message text source file shall be ignored. The effects of lines starting with any
18163 character other than those defined above are implementation-defined.

18164 Text strings can contain the special characters and escape sequences defined in the following
18165 table:

Description	Symbol	Sequence
<newline>	NL(LF)	\n
Horizontal tab	HT	\t
<vertical-tab>	VT	\v
<backspace>	BS	\b
<carriage-return>	CR	\r
<form-feed>	FF	\f
Backslash	\	\\
Bit pattern	ddd	\ddd

The escape sequence "\ddd" consists of backslash followed by one, two, or three octal digits, which shall be taken to specify the value of the desired character. If the character following a backslash is not one of those specified, the backslash shall be ignored.

Backslash ('\') followed by a <newline> is also used to continue a string on the following line. Thus, the following two lines describe a single message string:

```
1 This line continues \
to the next line
```

which shall be equivalent to:

```
1 This line continues to the next line
```

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Message catalogs produced by *gencat* are binary encoded, meaning that their portability cannot be guaranteed between different types of machine. Thus, just as C programs need to be recompiled for each type of machine, so message catalogs must be recreated via *gencat*.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iconv, the System Interfaces volume of IEEE Std 1003.1-200x, **<limits.h>**

CHANGE HISTORY

First released in Issue 3.

Issue 6

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

18207 **NAME**

18208 get — get a version of an SCCS file (**DEVELOPMENT**)

18209 **SYNOPSIS**

18210 XSI `get [–begkmnlLpst][–c cutoff][–i list][–r SID][–x list] file...`

18211

18212 **DESCRIPTION**

18213 The *get* utility shall generate a text file from each named SCCS *file* according to the specifications
18214 given by its options.

18215 The generated text shall normally be written into a file called the **g-file** whose name is derived
18216 from the SCCS filename by simply removing the leading "`s.`".

18217 **OPTIONS**

18218 The *get* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
18219 Utility Syntax Guidelines.

18220 The following options shall be supported:

18221 **–r** *SID* Indicate the SCCS Identification String (SID) of the version (delta) of an SCCS file
18222 to be retrieved. The table shows, for the most useful cases, what version of an
18223 SCCS file is retrieved (as well as the SID of the version to be eventually created by
18224 *delta* if the **–e** option is also used), as a function of the SID specified.

18225 **–c** *cutoff* Indicate the *cutoff* date-time, in the form:

18226 *YY*[*MM*[*DD*[*HH*[*MM*[*SS*]]]]]

18227 For the *YY* component, values in the range [69,99] shall refer to years 1969 to 1999
18228 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive.

18229 **Note:** It is expected that in a future version of IEEE Std 1003.1-200x the default
18230 century inferred from a 2-digit year will change. (This would apply to all
18231 commands accepting a 2-digit year as input.)

18232 No changes (deltas) to the SCCS file that were created after the specified *cutoff*
18233 date-time shall be included in the generated text file. Units omitted from the date-
18234 time default to their maximum possible values; for example, **–c** 7502 is equivalent
18235 to **–c** 750228235959.

18236 Any number of non-numeric characters may separate the various 2-digit pieces of
18237 the *cutoff* date-time. This feature allows the user to specify a *cutoff* date in the form:
18238 **–c** "77/2/2 9:22:25".

18239 **–e** Indicate that the *get* is for the purpose of editing or making a change (delta) to the
18240 SCCS file via a subsequent use of *delta*. The **–e** option used in a *get* for a particular
18241 version (SID) of the SCCS file shall prevent further *get* commands from editing on
18242 the same SID until *delta* is executed or the **j** (joint edit) flag is set in the SCCS file.
18243 Concurrent use of *get* **–e** for different SIDs is always allowed.

18244 If the **g-file** generated by *get* with a **–e** option is accidentally ruined in the process
18245 of editing, it may be regenerated by re-executing the *get* command with the **–k**
18246 option in place of the **–e** option.

18247 SCCS file protection specified via the ceiling, floor, and authorized user list stored
18248 in the SCCS file shall be enforced when the **–e** option is used.

18249 **–b** Use with the **–e** option to indicate that the new delta should have an SID in a new
18250 branch as shown in the table below. This option shall be ignored if the **b** flag is not
18251 present in the file or if the retrieved delta is not a leaf delta. (A leaf delta is one that

has no successors on the SCCS file tree.)

Note: A branch delta may always be created from a non-leaf delta.

–i *list* Indicate a *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

```
<list> ::= <range> | <list> , <range>
<range> ::= SID | SID – SID
```

SID, the SCCS Identification of a delta, may be in any form shown in the ''SID Specified'' column of the table in the EXTENDED DESCRIPTION section, except that the result of supplying a partial SID is unspecified. A diagnostic message shall be written if the first SID in the range is not an ancestor of the second SID in the range.

–x *list* Indicate a *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the **–i** option for the *list* format.

–k Suppress replacement of identification keywords (see below) in the retrieved text by their value. The **–k** option shall be implied by the **–e** option.

–l Write a delta summary into an **l-file**.

–L Write a delta summary to standard output. All informative output that normally is written to standard output shall be written to standard error instead, unless the **–s** option is used, in which case it shall be suppressed.

–p Write the text retrieved from the SCCS file to the standard output. No **g-file** shall be created. All informative output that normally goes to the standard output shall go to standard error instead, unless the **–s** option is used, in which case it shall disappear.

–s Suppress all informative output normally written to standard output. However, fatal error messages (which shall always be written to the standard error) shall remain unaffected.

–m Precede each text line retrieved from the SCCS file by the SID of the delta that inserted the text line in the SCCS file. The format shall be:

```
"%s\t%s", <SID>, <text line>
```

–n Precede each generated text line with the %**M**% identification keyword value (see below). The format shall be:

```
"%s\t%s", <%M% value>, <text line>
```

When both the **–m** and **–n** options are used, the <*text line*> shall be replaced by the **–m** option-generated format.

–g Suppress the actual retrieval of text from the SCCS file. It is primarily used to generate an **l-file**, or to verify the existence of a particular SID.

–t Use to access the most recently created (top) delta in a given release (for example, **–r 1**), or release and level (for example, **–r 1.2**).

OPERANDS

The following operands shall be supported:

file A pathname of an existing SCCS file or a directory. If *file* is a directory, the *get* utility shall behave as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin

with **s.**) and unreadable files shall be silently ignored.

If exactly one *file* operand appears, and it is `'–'`, the standard input shall be read; each line of the standard input is taken to be the name of an SCCS file to be processed. Non-SCCS files and unreadable files shall be silently ignored.

STDIN

The standard input shall be a text file used only if the *file* operand is specified as `'–'`. Each line of the text file shall be interpreted as an SCCS pathname.

INPUT FILES

The SCCS files shall be files of an unspecified format.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *get*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error, and informative messages written to standard output (or standard error, if the **–p** option is used).

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

TZ Determine the timezone in which the times and dates are written in the SCCS file are evaluated. If the *TZ* variable is unset or NULL, an unspecified system default timezone is used.

ASYNCHRONOUS EVENTS

Default.

STDOUT

For each file processed, *get* shall write to standard output the SID being accessed and the number of lines retrieved from the SCCS file, in the following format:

`"%s\n%d lines\n", <SID>, <number of lines>`

If the **–e** option is used, the SID of the delta to be made shall appear after the SID accessed and before the number of lines generated, in the POSIX locale:

```
"%s\nnew delta %s\n%d lines\n", <SID accessed>,
    <SID to be made>, <number of lines>
```

If there is more than one named file or if a directory or standard input is named, each pathname shall be written before each of the lines shown in one of the preceding formats:

`"\n%s:\n", <pathname>`

If the **–L** option is used, a delta summary shall be written following the format specified below for **l-files**.

If the **–i** option is used, included deltas shall be listed following the notation, in the POSIX locale:

```
"Included:\n"
```

If the **–x** option is used, excluded deltas shall be listed following the notation, in the POSIX locale:

```
"Excluded:\n"
```

If the **–p** or **–L** options are specified, the standard output shall consist of the text retrieved from the SCCS file.

STDERR

The standard error shall be used only for diagnostic messages, except if the **–p** or **–L** options are specified, it shall include all informative messages normally sent to standard output.

OUTPUT FILES

Several auxiliary files may be created by *get*. These files are known generically as the **g-file**, **l-file**, **p-file**, and **z-file**. The letter before the hyphen is called the *tag*. An auxiliary filename shall be formed from the SCCS filename: the application shall ensure that the last component of all SCCS filenames is of the form **s.***module-name*; the auxiliary files shall be named by replacing the leading **s** with the tag. The **g-file** shall be an exception to this scheme: the **g-file** is named by removing the **s.** prefix. For example, for **s.xyz.c**, the auxiliary filenames would be **xyz.c**, **l.xyz.c**, **p.xyz.c**, and **z.xyz.c**, respectively.

The **g-file**, which contains the generated text, shall be created in the current directory (unless the **–p** option is used). A **g-file** shall be created in all cases, whether or not any lines of text were generated by the *get*. It shall be owned by the real user. If the **–k** option is used or implied, the **g-file** shall be writable by the owner only (read-only for everyone else); otherwise, it shall be read-only. Only the real user need have write permission in the current directory.

The **l-file** shall contain a table showing which deltas were applied in generating the retrieved text. The **l-file** shall be created in the current directory if the **–l** option is used; it shall be read-only and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the **l-file** shall have the following format:

```
"%c%c%cΔ%s\t%sΔ%s\n", <code1>, <code2>, <code3>,
    <SID>, <date-time>, <login>
```

where the entries are:

<code1> A <space> if the delta was applied; `'*'` otherwise.

<code2> A <space> if the delta was applied or was not applied and ignored; `'*'` if the delta was not applied and was not ignored.

<code3> A character indicating a special reason why the delta was or was not applied:

I Included.

X Excluded.

C Cut off (by a **–c** option).

<date-time> Date and time (using the format of the *date* utility's `%y/%m/%d %T` conversion specification format) of creation.

<login> Login name of person who created *delta*.

The comments and MR data shall follow on subsequent lines, indented one <tab>. A blank line shall terminate each entry.

The **p-file** shall be used to pass information resulting from a *get* with a **–e** option along to *delta*. Its contents shall also be used to prevent a subsequent execution of *get* with a **–e** option for the same SID until *delta* is executed or the joint edit flag, **j**, is set in the SCCS file. The **p-file** shall be created in the directory containing the SCCS file and the application shall ensure that the effective user has write permission in that directory. It shall be writable by owner only, and owned by the effective user. Each line in the **p-file** shall have the following format:

```
"%sΔ%sΔ%sΔ%s%s%s\n", <g-file SID>,
    <SID of new delta>, <login-name of real user>,
    <date-time>, <i-value>, <x-value>
```

where <*i-value*> uses the format " " if no **–i** option was specified, and shall use the format:

```
"Δ–i%s", <–i option option-argument>
```

if a **–i** option was specified and <*x-value*> uses the format " " if no **–x** option was specified, and shall use the format:

```
"Δ–x%s", <–x option option-argument>
```

if a **–x** option was specified. There can be an arbitrary number of lines in the **p-file** at any time; no two lines shall have the same new delta SID.

The **z-file** shall serve as a lock-out mechanism against simultaneous updates. Its contents shall be the binary process ID of the command (that is, *get*) that created it. The **z-file** shall be created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the **p-file** shall apply for the **z-file**. The **z-file** shall be created read-only.

EXTENDED DESCRIPTION

Determination of SCCS Identification String				
SID* Specified	**–b Keyletter Used†**	**Other Conditions**	**SID Retrieved**	**SID of Delta to be Created**
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	–	R < mR and R does not exist	hR.mL**	hR.mL.(mB+1).1
R	–	Trunk successor in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk successor	R.L	R.(L+1)
R.L	yes	No trunk successor	R.L	R.L.(mB+1).1
R.L	–	Trunk successor in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch successor	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch successor	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch successor	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch successor	R.L.B.S	R.L.(mB+1).1
R.L.B.S	–	Branch successor	R.L.B.S	R.L.(mB+1).1

* R, L, B, and S are the release, level, branch, and sequence components of the SID, respectively; m means maximum. Thus, for example, R.mL means ‘‘the maximum level number within release R’’; R.L.(mB+1).1 means ‘‘the first sequence number on the new branch (that is, maximum branch number plus one) of level L within release R’’. Note that if the SID specified is of the form R.L, R.L.B, or R.L.B.S, each of the specified components shall exist.

** hR is the highest existing release that is lower than the specified, nonexistent, release R.

*** This is used to force creation of the first delta in a new release.

† The **–b** option is effective only if the **b** flag is present in the file. An entry of ’–’ means ‘‘irrelevant’’.

‡ This case applies if the **d** (default SID) flag is not present in the file. If the **d** flag is present in the file, then the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

System Date and Time

When a **g-file** is generated, the creation time of deltas in the SCCS file may be taken into account. If any of these times are apparently in the future, the behavior is unspecified.

Identification Keywords

Identifying information shall be inserted into the text retrieved from the SCCS file by replacing identification keywords with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

%**M**%	Module name: either the value of the **m** flag in the file, or if absent, the name of the SCCS file with the leading **s.** removed.
%**I**%	SCCS identification (SID) (%**R**%.%**L**% or %**R**%.%**L**%.%**B**%.%**S**%) of the retrieved text.
%**R**%	Release.
%**L**%	Level.
%**B**%	Branch.
%**S**%	Sequence.
%**D**%	Current date (*YY*/*MM*/*DD*).
%**H**%	Current date (*MM*/*DD*/*YY*).
%**T**%	Current time (*HH*:*MM*:*SS*).
%**E**%	Date newest applied delta was created (*YY*/*MM*/*DD*).
%**G**%	Date newest applied delta was created (*MM*/*DD*/*YY*).
%**U**%	Time newest applied delta was created (*HH*:*MM*:*SS*).
%**Y**%	Module type: value of the **t** flag in the SCCS file.
%**F**%	SCCS filename.
%**P**%	SCCS absolute pathname.
%**Q**%	The value of the **q** flag in the file.
%**C**%	Current line number. This keyword is intended for identifying messages output by the program, such as ‘‘this should not have happened’’ type errors. It is not intended to be used on every line to provide sequence numbers.
%**Z**%	The four-character string `"@(#)"` recognizable by *what*.
%**W**%	A shorthand notation for constructing *what* strings:
	`%W%=%Z%%M%<tab>%I%`
%**A**%	Another shorthand notation for constructing *what* strings:
	`%A%=%Z%%Y%%M%%I%%Z%`

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Problems can arise if the system date and time have been modified (for example, put forward and then back again, or unsynchronized clocks across a network) and can also arise when different values of the *TZ* environment variable are used.

Problems of a similar nature can also arise for the operation of the *delta* utility, which compares the previous file body against the working file as part of its normal operation.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

The **–lp** option may be withdrawn in a future issue.

SEE ALSO

admin, *delta*, *prs*, *what*

CHANGE HISTORY

First released in Issue 2.

Issue 5

Correction to the first format string in STDOUT.

The interpretation of the *YY* component of the **–c** *cutoff* argument is noted.

Issue 6

The obsolescent SYNOPSIS is removed, removing the **–lp** option.

The Open Group Corrigendum U025/5 is applied, correcting text in the OPTIONS section.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

The normative text is reworded to emphasize the term ‘‘shall’’ for implementation requirements.

The Open Group Corrigendum U048/1 is applied.

The EXTENDED DESCRIPTION section is updated to make partial SID handling unspecified, reflecting common usage, and to clarify SID ranges as per The Open Group Base Resolution bwg2001-007.

The Open Group Interpretation PIN4C.00014 is applied.

New text is added to the EXTENDED DESCRIPTION and APPLICATION USAGE sections regarding how the system date and time may be taken into account, and the *TZ* environment variable is added to the ENVIRONMENT VARIABLES section as per The Open Group Base Resolution bwg2001-007.

18509 **NAME**

18510 getconf — get configuration values

18511 **SYNOPSIS**

18512 `getconf [ –v specification ] system_var`

18513 `getconf [ –v specification ] path_var pathname`

18514 **DESCRIPTION**

18515 In the first synopsis form, the *getconf* utility shall write to the standard output the value of the
18516 variable specified by the *system_var* operand.

18517 In the second synopsis form, the *getconf* utility shall write to the standard output the value of the
18518 variable specified by the *path_var* operand for the path specified by the *pathname* operand.

18519 The value of each configuration variable shall be determined as if it were obtained by calling the
18520 function from which it is defined to be available by this volume of IEEE Std 1003.1-200x or by the
18521 System Interfaces volume of IEEE Std 1003.1-200x (see the OPERANDS section). The value shall
18522 reflect conditions in the current operating environment.

18523 **OPTIONS**

18524 The *getconf* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
18525 12.2, Utility Syntax Guidelines.

18526 The following option shall be supported:

18527 **–v** *specification*

18528 Indicate a specific specification and version for which configuration variables shall
18529 be determined. If this option is not specified, the values returned correspond to an
18530 implementation default conforming compilation environment.

18531 If the command:

18532 `getconf _POSIX_V6_ILP32_OFF32`

18533 does not write "–1\n" or "undefined\n" to standard output, then commands of
18534 the form:

18535 `getconf –v POSIX_V6_ILP32_OFF32 ...`

18536 determine values for configuration variables corresponding to the
18537 POSIX_V6_ILP32_OFF32 compilation environment specified in *c99* (on page 2413),
18538 EXTENDED DESCRIPTION.

18539 If the command:

18540 `getconf _POSIX_V6_ILP32_OFFBIG`

18541 does not write "–1\n" or "undefined\n" to standard output, then commands of
18542 the form:

18543 `getconf –v POSIX_V6_ILP32_OFFBIG ...`

18544 determine values for configuration variables corresponding to the
18545 POSIX_V6_ILP32_OFFBIG compilation environment specified in *c99* (on page
18546 2413), EXTENDED DESCRIPTION.

18547 If the command:

18548 `getconf _POSIX_V6_LP64_OFF64`

18549 does not write "–1\n" or "undefined\n" to standard output, then commands of
18550 the form:

```
getconf –v POSIX_V6_LP64_OFF64 ...
```

determine values for configuration variables corresponding to the POSIX_V6_LP64_OFF64 compilation environment specified in *c99* (on page 2413), EXTENDED DESCRIPTION.

If the command:

```
getconf _POSIX_V6_LPBIG_OFFBIG
```

does not write `"–1\n"` or `"undefined\n"` to standard output, then commands of the form:

```
getconf –v POSIX_V6_LPBIG_OFFBIG ...
```

determine values for configuration variables corresponding to the POSIX_V6_LPBIG_OFFBIG compilation environment specified in *c99* (on page 2413), EXTENDED DESCRIPTION.

OPERANDS

The following operands shall be supported:

path_var A name of a configuration variable. All of the variables in the *pathconf*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x are supported and the implementation may add other local variables.

pathname A pathname for which the variable specified by *path_var* is to be determined.

system_var A name of a configuration variable. All of the variables in the *confstr*( ) and *sysconf*( ) functions defined in the System Interfaces volume of IEEE Std 1003.1-200x shall be supported and the implementation may add other local values.

When the symbol listed in the first column of the following table is used as the *system_var* operand, *getconf* yields the same value as *confstr*( ) when called with the value in the second column:

system_var	*confstr*( ) Name Value
PATH	_CS_PATH
POSIX_V6_ILP32_OFF32_CFLAGS	_CS_POSIX_V6_ILP32_OFF32_CFLAGS
POSIX_V6_ILP32_OFF32_LDFLAGS	_CS_POSIX_V6_ILP32_OFF32_LDFLAGS
POSIX_V6_ILP32_OFF32_LIBS	_CS_POSIX_V6_ILP32_OFF32_LIBS
POSIX_V6_ILP32_OFF32_LINTFLAGS	_CS_POSIX_V6_ILP32_OFF32_LINTFLAGS
POSIX_V6_ILP32_OFFBIG_CFLAGS	_CS_POSIX_V6_ILP32_OFFBIG_CFLAGS
POSIX_V6_ILP32_OFFBIG_LDFLAGS	_CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS
POSIX_V6_ILP32_OFFBIG_LIBS	_CS_POSIX_V6_ILP32_OFFBIG_LIBS
POSIX_V6_ILP32_OFFBIG_LINTFLAGS	_CS_POSIX_V6_ILPBIG_OFF32_LINTFLAGS
POSIX_V6_LP64_OFF64_CFLAGS	_CS_POSIX_V6_LP64_OFF64_CFLAGS
POSIX_V6_LP64_OFF64_LDFLAGS	_CS_POSIX_V6_LP64_OFF64_LDFLAGS
POSIX_V6_LP64_OFF64_LIBS	_CS_POSIX_V6_LP64_OFF64_LIBS
POSIX_V6_LP64_OFF64_LINTFLAGS	_CS_POSIX_V6_LP64_OFF64_LINTFLAGS
POSIX_V6_LPBIG_OFFBIG_CFLAGS	_CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS
POSIX_V6_LPBIG_OFFBIG_LDFLAGS	_CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS
POSIX_V6_LPBIG_OFFBIG_LIBS	_CS_POSIX_V6_LPBIG_OFFBIG_LIBS

system_var	*confstr*() Name Value
POSIX_V6_LPBIG_OFFBIG_LINTFLAGS	_CS_POSIX_V6_LPBIG_OFFBIG_LINTFLAGS
POSIX_V6_WIDTH_RESTRICTED_ENVS	_CS_POSIX_V6_WIDTH_RESTRICTED_ENVS
XBS5_ILP32_OFF32_CFLAGS (**LEGACY**)	_CS_XBS5_ILP32_OFF32_CFLAGS
XBS5_ILP32_OFF32_LDFLAGS (**LEGACY**)	_CS_XBS5_ILP32_OFF32_LDFLAGS
XBS5_ILP32_OFF32_LIBS (**LEGACY**)	_CS_XBS5_ILP32_OFF32_LIBS
XBS5_ILP32_OFF32_LINTFLAGS (**LEGACY**)	_CS_XBS5_ILP32_OFF32_LINTFLAGS
XBS5_ILP32_OFFBIG_CFLAGS (**LEGACY**)	_CS_XBS5_ILP32_OFFBIG_CFLAGS
XBS5_ILP32_OFFBIG_LDFLAGS (**LEGACY**)	_CS_XBS5_ILP32_OFFBIG_LDFLAGS
XBS5_ILP32_OFFBIG_LIBS (**LEGACY**)	_CS_XBS5_ILP32_OFFBIG_LIBS
XBS5_ILP32_OFFBIG_LINTFLAGS (**LEGACY**)	_CS_XBS5_ILPBIG_OFF32_LINTFLAGS
XBS5_LP64_OFF64_CFLAGS (**LEGACY**)	_CS_XBS5_LP64_OFF64_CFLAGS
XBS5_LP64_OFF64_LDFLAGS (**LEGACY**)	_CS_XBS5_LP64_OFF64_LDFLAGS
XBS5_LP64_OFF64_LIBS (**LEGACY**)	_CS_XBS5_LP64_OFF64_LIBS
XBS5_LP64_OFF64_LINTFLAGS (**LEGACY**)	_CS_XBS5_LP64_OFF64_LINTFLAGS
XBS5_LPBIG_OFFBIG_CFLAGS (**LEGACY**)	_CS_XBS5_LPBIG_OFFBIG_CFLAGS
XBS5_LPBIG_OFFBIG_LDFLAGS (**LEGACY**)	_CS_XBS5_LPBIG_OFFBIG_LDFLAGS
XBS5_LPBIG_OFFBIG_LIBS (**LEGACY**)	_CS_XBS5_LPBIG_OFFBIG_LIBS
XBS5_LPBIG_OFFBIG_LINTFLAGS (**LEGACY**)	_CS_XBS5_LPBIG_OFFBIG_LINTFLAGS

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *getconf*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If the specified variable is defined on the system and its value is described to be available from the *confstr*() function defined in the System Interfaces volume of IEEE Std 1003.1-200x, its value shall be written in the following format:

```
"%s\n", <value>
```

Otherwise, if the specified variable is defined on the system, its value shall be written in the following format:

```
"%d\n", <value>
```

If the specified variable is valid, but is undefined on the system, *getconf* shall write using the following format:

```
"undefined\n"
```

If the variable name is invalid or an error occurs, nothing shall be written to standard output.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 The specified variable is valid and information about its current state was written successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

The following example illustrates the value of {NGROUPS_MAX}:

```
getconf NGROUPS_MAX
```

The following example illustrates the value of {NAME_MAX} for a specific directory:

```
getconf NAME_MAX /usr
```

The following example shows how to deal more carefully with results that might be unspecified:

```
if value=$(getconf PATH_MAX /usr); then
    if [ "$value" = "undefined" ]; then
        echo PATH_MAX in /usr is infinite.
    else
        echo PATH_MAX in /usr is $value.
    fi
else
    echo Error in getconf.
fi
```

Note that:

```
sysconf(_SC_POSIX_C_BIND);
```

and:

```
system("getconf POSIX2_C_BIND");
```

in a C program could give different answers. The *sysconf*( ) call supplies a value that corresponds to the conditions when the program was either compiled or executed, depending on the implementation; the *system*( ) call to *getconf* always supplies a value corresponding to conditions when the program is executed.

RATIONALE

The original need for this utility, and for the *confstr*( ) function, was to provide a way of finding the configuration-defined default value for the *PATH* environment variable. Since *PATH* can be modified by the user to include directories that could contain utilities replacing the standard utilities, shell scripts need a way to determine the system-supplied *PATH* environment variable value that contains the correct search path for the standard utilities. It was later suggested that access to the other variables described in this volume of IEEE Std 1003.1-200x could also be useful to applications.

This functionality of *getconf* would not be adequately subsumed by another command such as:

```
grep var /etc/conf
```

because such a strategy would provide correct values for neither those variables that can vary at runtime, nor those that can vary depending on the path.

Early proposal versions of *getconf* specified exit status 1 when the specified variable was valid, but not defined on the system. The output string `"undefined"` is now used to specify this case with exit code 0 because so many things depend on an exit code of zero when an invoked utility is successful.

FUTURE DIRECTIONS

None.

SEE ALSO

c99, the System Interfaces volume of IEEE Std 1003.1-200x, *confstr*( ), *pathconf*( ), *sysconf*( )

CHANGE HISTORY

First released in Issue 4.

Issue 5

In the OPERANDS section:

- {NL_MAX} is changed to {NL_NMAX}.
- Entries beginning NL_ are deleted from the list of standard configuration variables.
- The list of variables previously marked UX is merged with the list marked EX.
- Operands are added to support new Option Groups.
- Operands are added so that *getconf* can determine supported programming environments.

Issue 6

The Open Group Corrigendum U029/4 is applied, correcting the example command in the last paragraph of the OPTIONS section.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- Operands are added to determine supported programming environments.

This reference page is updated for alignment with the ISO/IEC 9899:1999 standard. Specifically, new macros for *c99* programming environments are introduced.

18722 XSI marked *system_var* (XBS5_*) values are marked LEGACY.

18723 **NAME**

18724 getopts — parse utility options

18725 **SYNOPSIS**

18726 `getopts` *`optstring name`* `[`*`arg...`*`]`

18727 **DESCRIPTION**

18728 The *getopts* utility shall retrieve options and option-arguments from a list of parameters. It shall
18729 support the Utility Syntax Guidelines 3 to 10, inclusive, described in the Base Definitions volume
18730 of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

18731 Each time it is invoked, the *getopts* utility shall place the value of the next option in the shell
18732 variable specified by the *name* operand and the index of the next argument to be processed in the
18733 shell variable *OPTIND*. Whenever the shell is invoked, *OPTIND* shall be initialized to 1.

18734 When the option requires an option-argument, the *getopts* utility shall place it in the shell
18735 variable *OPTARG*. If no option was found, or if the option that was found does not have an
18736 option-argument, *OPTARG* shall be unset.

18737 If an option character not contained in the *optstring* operand is found where an option character
18738 is expected, the shell variable specified by *name* shall be set to the question-mark (`'?'`) character.
18739 In this case, if the first character in *optstring* is a colon (`':'`), the shell variable *OPTARG* shall be
18740 set to the option character found, but no output shall be written to standard error; otherwise, the
18741 shell variable *OPTARG* shall be unset and a diagnostic message shall be written to standard
18742 error. This condition shall be considered to be an error detected in the way arguments were
18743 presented to the invoking application, but shall be not an error in *getopts* processing.

18744 If an option-argument is missing:

18745 • If the first character of *optstring* is a colon, the shell variable specified by *name* shall be set to
18746 the colon character and the shell variable *OPTARG* shall be set to the option character found.

18747 • Otherwise, the shell variable specified by *name* shall be set to the question-mark character,
18748 the shell variable *OPTARG* shall be unset, and a diagnostic message shall be written to
18749 standard error. This condition shall be considered to be an error detected in the way
18750 arguments were presented to the invoking application, but shall not be an error in *getopts*
18751 processing; a diagnostic message shall be written as stated, but the exit status shall be zero.

18752 When the end of options is encountered, the *getopts* utility shall exit with a return value greater
18753 than zero; the shell variable *OPTIND* shall be set to the index of the first non-option-argument,
18754 where the first `"--"` argument is considered to be an option-argument if there are no other
18755 non-option-arguments appearing before it, or the value `"$#"+1` if there are no non-option-
18756 arguments; the *name* variable shall be set to the question-mark character. Any of the following
18757 shall identify the end of options: the special option `"--"`, finding an argument that does not
18758 begin with a `'-'`, or encountering an error.

18759 The shell variables *OPTIND* and *OPTARG* shall be local to the caller of *getopts* and shall not be
18760 exported by default.

18761 The shell variable specified by the *name* operand, *OPTIND* and *OPTARG* shall affect the current
18762 shell execution environment; see Section 2.12 (on page 2263).

18763 If the application sets *OPTIND* to the value 1, a new set of parameters can be used: either the
18764 current positional parameters or new *arg* values. Any other attempt to invoke *getopts* multiple
18765 times in a single shell execution environment with parameters (positional parameters or *arg*
18766 operands) that are not the same in all invocations, or with an *OPTIND* value modified to be a
18767 value other than 1, produces unspecified results.

OPTIONS

None.

OPERANDS

The following operands shall be supported:

optstring A string containing the option characters recognized by the utility invoking *getopts*. If a character is followed by a colon, the option shall be expected to have an argument, which should be supplied as a separate argument. Applications should specify an option character and its option-argument as separate arguments, but *getopts* shall interpret the characters following an option character requiring arguments as an argument whether or not this is done. An explicit null option-argument need not be recognized if it is not supplied as a separate argument when *getopts* is invoked. (See also the *getopt*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x.) The characters question-mark and colon shall not be used as option characters by an application. The use of other option characters that are not alphanumeric produces unspecified results. If the option-argument is not supplied as a separate argument from the option character, the value in *OPTARG* shall be stripped of the option character and the `'-'`. The first character in *optstring* determines how *getopts* behaves if an option character is not known or an option-argument is missing.

name The name of a shell variable that shall be set by the *getopts* utility to the option character that was found.

The *getopts* utility by default shall parse positional parameters passed to the invoking shell procedure. If *arg*s are given, they shall be parsed instead of the positional parameters.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *getopts*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

OPTIND This variable shall be used by the *getopts* utility as the index of the next argument to be processed.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

Whenever an error is detected and the first character in the *optstring* operand is not a colon (':'), a diagnostic message shall be written to standard error with the following information in an unspecified format:

- The invoking program name shall be identified in the message. The invoking program name shall be the value of the shell special parameter 0 (see Section 2.5.2 (on page 2235)) at the time the *getopts* utility is invoked. A name equivalent to:

  ```
  basename "$0"
  ```

  may be used.

- If an option is found that was not specified in *optstring*, this error is identified and the invalid option character shall be identified in the message.

- If an option requiring an option-argument is found, but an option-argument is not found, this error shall be identified and the invalid option character shall be identified in the message.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 An option, specified or unspecified by *optstring*, was found.

>0 The end of options was encountered or an error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Since *getopts* affects the current shell execution environment, it is generally provided as a shell regular built-in. If it is called in a subshell or separate utility execution environment, such as one of the following:

```
(getopts abc value "$@")
nohup getopts ...
find . –exec getopts ... \;
```

it does not affect the shell variables in the caller's environment.

Note that shell functions share *OPTIND* with the calling shell even though the positional parameters are changed. If the calling shell and any of its functions uses *getopts* to parse arguments, the results are unspecified.

EXAMPLES

The following example script parses and displays its arguments:

```
aflag=
bflag=
```

```
    while getopts ab: name
    do
        case $name in
        a)    aflag=1;;
        b)    bflag=1
              bval="$OPTARG";;
        ?)   printf "Usage: %s: [–a] [–b value] args\n" $0
              exit 2;;
        esac
    done
    if [ ! –z "$aflag" ]; then
        printf "Option –a specified\n"
    fi
    if [ ! –z "$bflag" ]; then
        printf 'Option –b "%s" specified\n' "$bval"
    fi
    shift $(($OPTIND – 1))
    printf "Remaining arguments are: %s\n" "$*"
```

RATIONALE

The *getopts* utility was chosen in preference to the System V *getopt* utility because *getopts* handles option-arguments containing <blank>s.

The *OPTARG* variable is not mentioned in the ENVIRONMENT VARIABLES section because it does not affect the execution of *getopts*; it is one of the few ‘‘output-only’’ variables used by the standard utilities.

The colon is not allowed as an option character because that is not historical behavior, and it violates the Utility Syntax Guidelines. The colon is now specified to behave as in the KornShell version of the *getopts* utility; when used as the first character in the *optstring* operand, it disables diagnostics concerning missing option-arguments and unexpected option characters. This replaces the use of the *OPTERR* variable that was specified in an early proposal.

The formats of the diagnostic messages produced by the *getopts* utility and the *getopt*( ) function are not fully specified because implementations with superior (‘‘friendlier’’) formats objected to the formats used by some historical implementations. The standard developers considered it important that the information in the messages used be uniform between *getopts* and *getopt*( ). Exact duplication of the messages might not be possible, particularly if a utility is built on another system that has a different *getopt*( ) function, but the messages must have specific information included so that the program name, invalid option character, and type of error can be distinguished by a user.

Only a rare application program intercepts a *getopts* standard error message and wants to parse it. Therefore, implementations are free to choose the most usable messages they can devise. The following formats are used by many historical implementations:

```
"%s: illegal option –– %c\n", <program name>, <option character>
```

```
"%s: option requires an argument –– %c\n", <program name>, \
    <option character>
```

Historical shells with built-in versions of *getopt*( ) or *getopts* have used different formats, frequently not even indicating the option character found in error.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *getopt*( )

CHANGE HISTORY

First released in Issue 4.

Issue 6

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

18908 **NAME**

18909 grep — search a file for a pattern

18910 **SYNOPSIS**

18911 `grep [–E| –F][–c| –l| –q][–insvx] –e pattern_list...`
18912 `[–f pattern_file]...[file...]`

18913 `grep [–E| –F][–c| –l| –q][–insvx][–e pattern_list]...`
18914 `–f pattern_file...[file...]`

18915 `grep [–E| –F][–c| –l| –q][–insvx] pattern_list[file...]`

18916 **DESCRIPTION**

18917 The *grep* utility shall search the input files, selecting lines matching one or more patterns; the
18918 types of patterns are controlled by the options specified. The patterns are specified by the **–e**
18919 option, **–f** option, or the *pattern_list* operand. The *pattern_list*'s value shall consist of one or more
18920 patterns separated by <newline>s; the *pattern_file*'s contents shall consist of one or more
18921 patterns terminated by <newline>. By default, an input line shall be selected if any pattern,
18922 treated as an entire basic regular expression (BRE) as described in the Base Definitions volume of
18923 IEEE Std 1003.1-200x, Section 9.3, Basic Regular Expressions, matches any part of the line
18924 excluding the terminating <newline>; a null BRE shall match every line. By default, each selected
18925 input line shall be written to the standard output.

18926 Regular expression matching shall be based on text lines. Since a <newline> separates or
18927 terminates patterns (see the **–e** and **–f** options below), regular expressions cannot contain a
18928 <newline>. Similarly, since patterns are matched against individual lines (excluding the
18929 terminating <newline>s) of the input, there is no way for a pattern to match a <newline> found
18930 in the input.

18931 **OPTIONS**

18932 The *grep* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
18933 12.2, Utility Syntax Guidelines.

18934 The following options shall be supported:

18935 **–E** Match using extended regular expressions. Treat each pattern specified as an ERE,
18936 as described in the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.4,
18937 Extended Regular Expressions. If any entire ERE pattern matches some part of an
18938 input line excluding the terminating <newline>, the line shall be matched. A null
18939 ERE shall match every line.

18940 **–F** Match using fixed strings. Treat each pattern specified as a string instead of a
18941 regular expression. If an input line contains any of the patterns as a contiguous
18942 sequence of bytes, the line shall be matched. A null string shall match every line.

18943 **–c** Write only a count of selected lines to standard output.

18944 **–e** *pattern_list*
18945 Specify one or more patterns to be used during the search for input. The
18946 application shall ensure that patterns in *pattern_list* are separated by a <newline>.
18947 A null pattern can be specified by two adjacent <newline>s in *pattern_list*. Unless
18948 the **–E** or **–F** option is also specified, each pattern shall be treated as a BRE, as
18949 described in the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.3, Basic
18950 Regular Expressions. Multiple **–e** and **–f** options shall be accepted by the *grep*
18951 utility. All of the specified patterns shall be used when matching lines, but the
18952 order of evaluation is unspecified.

18953 **–f** *pattern_file*
18954 Read one or more patterns from the file named by the pathname *pattern_file*.
18955 Patterns in *pattern_file* shall be terminated by a <newline>. A null pattern can be
18956 specified by an empty line in *pattern_file*. Unless the **–E** or **–F** option is also
18957 specified, each pattern shall be treated as a BRE, as described in the Base
18958 Definitions volume of IEEE Std 1003.1-200x, Section 9.3, Basic Regular Expressions.

18959 **–i** Perform pattern matching in searches without regard to case; see the Base
18960 Definitions volume of IEEE Std 1003.1-200x, Section 9.2, Regular Expression
18961 General Requirements.

18962 **–l** (The letter ell.) Write only the names of files containing selected lines to standard
18963 output. Pathnames shall be written once per file searched. If the standard input is
18964 searched, a pathname of `"(standard input)"` shall be written, in the POSIX
18965 locale. In other locales, `"standard input"` may be replaced by something more
18966 appropriate in those locales.

18967 **–n** Precede each output line by its relative line number in the file, each file starting at
18968 line 1. The line number counter shall be reset for each file processed.

18969 **–q** Quiet. Nothing shall be written to the standard output, regardless of matching |
18970 lines. Exit with zero status if an input line is selected.

18971 **–s** Suppress the error messages ordinarily written for nonexistent or unreadable files.
18972 Other error messages shall not be suppressed.

18973 **–v** Select lines not matching any of the specified patterns. If the **–v** option is not
18974 specified, selected lines shall be those that match any of the specified patterns.

18975 **–x** Consider only input lines that use all characters in the line excluding the
18976 terminating <newline> to match an entire fixed string or regular expression to be
18977 matching lines.

18978 **OPERANDS**
18979 The following operands shall be supported:

18980 *pattern_list* Specify one or more patterns to be used during the search for input. This operand
18981 shall be treated as if it were specified as **–e** *pattern_list*.

18982 *file* A pathname of a file to be searched for the patterns. If no *file* operands are
18983 specified, the standard input shall be used.

18984 **STDIN**
18985 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES
18986 section.

18987 **INPUT FILES**
18988 The input files shall be text files.

18989 **ENVIRONMENT VARIABLES**
18990 The following environment variables shall affect the execution of *grep*:

18991 *LANG* Provide a default value for the internationalization variables that are unset or null.
18992 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
18993 Internationalization Variables for the precedence of internationalization variables
18994 used to determine the values of locale categories.)

18995 *LC_ALL* If set to a non-empty string value, override the values of all the other
18996 internationalization variables.

18997 *LC_COLLATE*
18998 Determine the locale for the behavior of ranges, equivalence classes and multi-
18999 character collating elements within regular expressions.

19000 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
19001 characters (for example, single-byte as opposed to multi-byte characters in
19002 arguments and input files) and the behavior of character classes within regular
19003 expressions.

19004 *LC_MESSAGES*
19005 Determine the locale that should be used to affect the format and contents of
19006 diagnostic messages written to standard error.

19007 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

19008 **ASYNCHRONOUS EVENTS**
19009 Default.

19010 **STDOUT**
19011 If the **–l** option is in effect, and the **–q** option is not, the following shall be written for each file
19012 containing at least one selected input line:

19013 `"%s\n", <file>`

19014 Otherwise, if more than one *file* argument appears, and **–q** is not specified, the *grep* utility shall
19015 prefix each output line by:

19016 `"%s:", <file>`

19017 The remainder of each output line shall depend on the other options specified:

19018 • If the **–c** option is in effect, the remainder of each output line shall contain:

19019 `"%d\n", <count>`

19020 • Otherwise, if **–c** is not in effect and the **–n** option is in effect, the following shall be written to
19021 standard output:

19022 `"%d:", <line number>`

19023 • Finally, the following shall be written to standard output:

19024 `"%s", <selected-line contents>`

19025 **STDERR**
19026 The standard error shall be used only for diagnostic messages.

19027 **OUTPUT FILES**
19028 None.

19029 **EXTENDED DESCRIPTION**
19030 None.

19031 **EXIT STATUS**
19032 The following exit values shall be returned:

19033 0 One or more lines were selected.

19034 1 No lines were selected.

19035 >1 An error occurred.

CONSEQUENCES OF ERRORS

If the **–q** option is specified, the exit status shall be zero if an input line is selected, even if an error was detected. Otherwise, default actions shall be performed.

APPLICATION USAGE

Care should be taken when using characters in *pattern_list* that may also be meaningful to the command interpreter. It is safest to enclose the entire *pattern_list* argument in single quotes:

```
'...'
```

The **–e** *pattern_list* option has the same effect as the *pattern_list* operand, but is useful when *pattern_list* begins with the hyphen delimiter. It is also useful when it is more convenient to provide multiple patterns as separate arguments.

Multiple **–e** and **–f** options are accepted and *grep* uses all of the patterns it is given while matching input text lines. (Note that the order of evaluation is not specified. If an implementation finds a null string as a pattern, it is allowed to use that pattern first, matching every line, and effectively ignore any other patterns.)

The **–q** option provides a means of easily determining whether or not a pattern (or string) exists in a group of files. When searching several files, it provides a performance improvement (because it can quit as soon as it finds the first match) and requires less care by the user in choosing the set of files to supply as arguments (because it exits zero if it finds a match even if *grep* detected an access or read error on earlier *file* operands).

EXAMPLES

1. To find all uses of the word "Posix" (in any case) in file **text.mm** and write with line numbers:

   ```
   grep –i –n posix text.mm
   ```

2. To find all empty lines in the standard input:

   ```
   grep ^$
   ```

   or:

   ```
   grep –v .
   ```

3. Both of the following commands print all lines containing strings "abc" or "def" or both:

   ```
   grep –E 'abc|def'
   grep –F 'abc|def'
   ```

4. Both of the following commands print all lines matching exactly "abc" or "def":

   ```
   grep –E '^abc$|^def$'
   grep –F –x 'abc|def'
   ```

RATIONALE

This *grep* has been enhanced in an upward-compatible way to provide the exact functionality of the historical *egrep* and *fgrep* commands as well. It was the clear intention of the standard developers to consolidate the three *grep*s into a single command.

The old *egrep* and *fgrep* commands are likely to be supported for many years to come as implementation extensions, allowing historical applications to operate unmodified.

Historical implementations usually silently ignored all but one of multiply-specified **–e** and **–f** options, but were not consistent as to which specification was actually used.

The –**b** option was omitted from the OPTIONS section because block numbers are implementation-defined.

The System V restriction on using – to mean standard input was omitted.

A definition of action taken when given a null BRE or ERE is specified. This is an error condition in some historical implementations.

The –**l** option previously indicated that its use was undefined when no files were explicitly named. This behavior was historical and placed an unnecessary restriction on future implementations. It has been removed.

The historical BSD *grep* –**s** option practice is easily duplicated by redirecting standard output to **/dev/null**. The –**s** option required here is from System V.

The –**x** option, historically available only with *fgrep*, is available here for all of the non-obsolescent versions.

FUTURE DIRECTIONS

None.

SEE ALSO

sed

CHANGE HISTORY

First released in Issue 2.

Issue 6

The Open Group Corrigendum U029/5 is applied, correcting the SYNOPSIS.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

19098 **NAME**

19099 hash — remember or report utility locations

19100 **SYNOPSIS**

19101 XSI
```
hash [utility...]
```

19102
```
hash –r
```

19103

19104 **DESCRIPTION**

19105 The *hash* utility shall affect the way the current shell environment remembers the locations of
19106 utilities found as described in Section 2.9.1.1 (on page 2249). Depending on the arguments
19107 specified, it shall add utility locations to its list of remembered locations or it shall purge the
19108 contents of the list. When no arguments are specified, it shall report on the contents of the list.

19109 Utilities provided as built-ins to the shell shall not be reported by *hash*.

19110 **OPTIONS**

19111 The *hash* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
19112 12.2, Utility Syntax Guidelines.

19113 The following option shall be supported:

19114 **–r** Forget all previously remembered utility locations.

19115 **OPERANDS**

19116 The following operand shall be supported:

19117 *utility* The name of a utility to be searched for and added to the list of remembered
19118 locations. If *utility* contains one or more slashes, the results are unspecified.

19119 **STDIN**

19120 Not used.

19121 **INPUT FILES**

19122 None.

19123 **ENVIRONMENT VARIABLES**

19124 The following environment variables shall affect the execution of *hash*:

19125 *LANG* Provide a default value for the internationalization variables that are unset or null.
19126 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
19127 Internationalization Variables for the precedence of internationalization variables
19128 used to determine the values of locale categories.)

19129 *LC_ALL* If set to a non-empty string value, override the values of all the other
19130 internationalization variables.

19131 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
19132 characters (for example, single-byte as opposed to multi-byte characters in
19133 arguments).

19134 *LC_MESSAGES*
19135 Determine the locale that should be used to affect the format and contents of
19136 diagnostic messages written to standard error.

19137 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

19138 *PATH* Determine the location of *utility*, as described in the Base Definitions volume of
19139 IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

19140 **ASYNCHRONOUS EVENTS**
19141 Default.

19142 **STDOUT**
19143 The standard output of *hash* shall be used when no arguments are specified. Its format is
19144 unspecified, but includes the pathname of each utility in the list of remembered locations for the
19145 current shell environment. This list shall consist of those utilities named in previous *hash*
19146 invocations that have been invoked, and may contain those invoked and found through the
19147 normal command search process.

19148 **STDERR**
19149 The standard error shall be used only for diagnostic messages. |

19150 **OUTPUT FILES**
19151 None.

19152 **EXTENDED DESCRIPTION**
19153 None.

19154 **EXIT STATUS**
19155 The following exit values shall be returned:

19156 0 Successful completion.

19157 >0 An error occurred.

19158 **CONSEQUENCES OF ERRORS**
19159 Default.

19160 **APPLICATION USAGE**
19161 Since *hash* affects the current shell execution environment, it is always provided as a shell
19162 regular built-in. If it is called in a separate utility execution environment, such as one of the
19163 following:

19164 `nohup hash -r`
19165 `find . -type f | xargs hash`

19166 it does not affect the command search process of the caller's environment.

19167 The *hash* utility may be implemented as an alias—for example, *alias* **–t –**, in which case utilities
19168 found through normal command search are not listed by the *hash* command.

19169 The effects of *hash* **–r** can also be achieved portably by resetting the value of *PATH*; in the
19170 simplest form, this can be:

19171 `PATH="$PATH"`

19172 The use of *hash* with *utility* names is unnecessary for most applications, but may provide a
19173 performance improvement on a few implementations; normally, the hashing process is included
19174 by default.

19175 **EXAMPLES**
19176 None.

19177 **RATIONALE**
19178 None.

19179 **FUTURE DIRECTIONS**
19180 None.

19181 **SEE ALSO**

19182 Section 2.9.1.1 (on page 2249)

19183 **CHANGE HISTORY**

19184 First released in Issue 2.

19185 **NAME**

19186 head — copy the first part of files

19187 **SYNOPSIS**

19188 `head [–n number][file...]`

19189 **DESCRIPTION**

19190 The *head* utility shall copy its input files to the standard output, ending the output for each file at
19191 a designated point.

19192 Copying shall end at the point in each input file indicated by the **–n** *number* option. The option-
19193 argument *number* shall be counted in units of lines.

19194 **OPTIONS**

19195 The *head* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
19196 12.2, Utility Syntax Guidelines.

19197 The following option shall be supported:

19198 **–n** *number* The first *number* lines of each input file shall be copied to standard output. The
19199 application shall ensure that the *number* option-argument is a positive decimal
19200 integer.

19201 If no options are specified, *head* shall act as if **–n 10** had been specified.

19202 **OPERANDS**

19203 The following operand shall be supported:

19204 *file* A pathname of an input file. If no *file* operands are specified, the standard input
19205 shall be used.

19206 **STDIN**

19207 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES
19208 section.

19209 **INPUT FILES**

19210 Input files shall be text files, but the line length is not restricted to {LINE_MAX} bytes.

19211 **ENVIRONMENT VARIABLES**

19212 The following environment variables shall affect the execution of *head*:

19213 *LANG* Provide a default value for the internationalization variables that are unset or null.
19214 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
19215 Internationalization Variables for the precedence of internationalization variables
19216 used to determine the values of locale categories.)

19217 *LC_ALL* If set to a non-empty string value, override the values of all the other
19218 internationalization variables.

19219 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
19220 characters (for example, single-byte as opposed to multi-byte characters in
19221 arguments and input files).

19222 *LC_MESSAGES*
19223 Determine the locale that should be used to affect the format and contents of
19224 diagnostic messages written to standard error.

19225 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

19226 **ASYNCHRONOUS EVENTS**
19227 Default.

19228 **STDOUT**
19229 The standard output shall contain designated portions of the input files.

19230 If multiple *file* operands are specified, *head* shall precede the output for each with the header:

19231 `"\n==> %s <==\n", <pathname>`

19232 except that the first header written shall not include the initial <newline>.

19233 **STDERR**
19234 The standard error shall be used only for diagnostic messages.

19235 **OUTPUT FILES**
19236 None.

19237 **EXTENDED DESCRIPTION**
19238 None.

19239 **EXIT STATUS**
19240 The following exit values shall be returned:

19241 0 Successful completion.

19242 >0 An error occurred.

19243 **CONSEQUENCES OF ERRORS**
19244 Default.

19245 **APPLICATION USAGE**
19246 The obsolescent *–number* form is withdrawn in this version. Applications should use the **–n**
19247 *number* option.

19248 **EXAMPLES**
19249 To write the first ten lines of all files (except those with a leading period) in the directory:

19250 ```
head *
```

19251 **RATIONALE**
19252 Although it is possible to simulate *head* with *sed* 10q for a single file, the standard developers
19253 decided that the popularity of *head* on historical BSD systems warranted its inclusion alongside
19254 *tail*.

19255 This standard version of *head* follows the Utility Syntax Guidelines. The **–n** option was added to
19256 this new interface so that *head* and *tail* would be more logically related.

19257 There is no **–c** option (as there is in *tail*) because it is not historical practice and because other
19258 utilities in this volume of IEEE Std 1003.1-200x provide similar functionality.

19259 **FUTURE DIRECTIONS**
19260 None.

19261 **SEE ALSO**
19262 *sed*, *tail*

19263 **CHANGE HISTORY**
19264 First released in Issue 4.
```

19265 **Issue 6**

19266 The obsolescent **–number** form is withdrawn.

19267 The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

19268 **NAME**

19269 iconv — codeset conversion

19270 **SYNOPSIS**

19271 `iconv [–cs] –f` *`fromcode`* `–t` *`tocode`* `[`*`file`* `...]`

19272 `iconv –l`

19273 **DESCRIPTION**

19274 The *iconv* utility shall convert the encoding of characters in *file* from one codeset to another and
19275 write the results to standard output.

19276 When the options indicate that charmap files are used to specify the codesets (see OPTIONS),
19277 the codeset conversion shall be accomplished by performing a logical join on the symbolic
19278 character names in the two charmaps. The implementation need not support the use of charmap
19279 files for codeset conversion unless the POSIX2_LOCALEDEF symbol is defined on the system.

19280 **OPTIONS**

19281 The *iconv* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
19282 12.2, Utility Syntax Guidelines.

19283 The following options shall be supported:

19284 **–c** Omit any invalid characters from the output. When **–c** is not used, the results of
19285 encountering invalid characters in the input stream (either those that are not valid
19286 members of the *fromcode* or those that have no corresponding value in *tocode*) shall
19287 be specified in the system documentation. The presence or absence of **–c** shall not
19288 affect the exit status of *iconv*.

19289 **–f** *fromcode* Identify the codeset of the input file. If the option-argument contains a slash
19290 character, *iconv* shall attempt to use it as the pathname of a charmap file, as
19291 defined in the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.4,
19292 Character Set Description File. If the pathname does not represent a valid, readable
19293 charmap file, the results are undefined. If the option-argument does not contain a
19294 slash, it shall be considered the name of one of the codeset descriptions provided
19295 by the system, in an unspecified format. The valid values of the option-argument
19296 without a slash are implementation-defined. If this option is omitted, the codeset
19297 of the current locale shall be used.

19298 **–l** Write all supported *fromcode* and *tocode* values to standard output in an unspecified
19299 format.

19300 **–s** Suppress any messages written to standard error concerning invalid characters.
19301 When **–s** is not used, the results of encountering invalid characters in the input
19302 stream (either those that are not valid members of the *fromcode* or those that have
19303 no corresponding value in *tocode*) shall be specified in the system documentation.
19304 The presence or absence of **–s** shall not affect the exit status of *iconv*.

19305 **–t** *tocode* Identify the codeset to be used for the output file. The semantics shall be
19306 equivalent to the **–f** *fromcode* option.

19307 If either **–f** or **–t** represents a charmap file, but the other does not (or is omitted), or both **–f** and
19308 **–t** are omitted, the results are undefined.

19309 **OPERANDS**

19310 The following operand shall be supported:

19311 *file* A pathname of an input file. If no *file* operands are specified, or if a *file* operand is
19312 `'–'`, the standard input shall be used.

19313 **STDIN**

19314 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '–'.

19315 **INPUT FILES**

19316 The input file shall be a text file.

19317 **ENVIRONMENT VARIABLES**

19318 The following environment variables shall affect the execution of *iconv*:

19319 *LANG* Provide a default value for the internationalization variables that are unset or null.
19320 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
19321 Internationalization Variables for the precedence of internationalization variables
19322 used to determine the values of locale categories.)

19323 *LC_ALL* If set to a non-empty string value, override the values of all the other
19324 internationalization variables.

19325 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
19326 characters (for example, single-byte as opposed to multi-byte characters in
19327 arguments). During translation of the file, this variable is superseded by the use of
19328 the *fromcode* option-argument.

19329 *LC_MESSAGES*
19330 Determine the locale that should be used to affect the format and contents of
19331 diagnostic messages written to standard error.

19332 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

19333 **ASYNCHRONOUS EVENTS**

19334 Default.

19335 **STDOUT**

19336 When the **–l** option is used, the standard output shall contain all supported *fromcode* and *tocode*
19337 values, written in an unspecified format.

19338 When the **–l** option is not used, the standard output shall contain the sequence of characters
19339 read from the input files, translated to the specified codeset. Nothing else shall be written to the
19340 standard output.

19341 **STDERR**

19342 The standard error shall be used only for diagnostic messages.

19343 **OUTPUT FILES**

19344 None.

19345 **EXTENDED DESCRIPTION**

19346 None.

19347 **EXIT STATUS**

19348 The following exit values shall be returned:

19349 0 Successful completion.

19350 >0 An error occurred.

19351 **CONSEQUENCES OF ERRORS**

19352 Default.

APPLICATION USAGE

The user must ensure that both charmap files use the same symbolic names for characters the two codesets have in common.

EXAMPLES

The following example converts the contents of file **mail.x400** from the ISO/IEC 6937:1994 standard codeset to the ISO/IEC 8859-1:1998 standard codeset, and stores the results in file **mail.local**:

```
iconv –f IS6937 –t IS8859 mail.x400 > mail.local
```

RATIONALE

The *iconv* utility can be used portably only when the user provides two charmap files as option-arguments. This is because a single charmap provided by the user cannot reliably be joined with the names in a system-provided character set description. The valid values for *fromcode* and *tocode* are implementation-defined and do not have to have any relation to the charmap mechanisms. As an aid to interactive users, the **–l** option was adopted from the Plan 9 operating system. It writes information concerning these implementation-defined values. The format is unspecified because there are many possible useful formats that could be chosen, such as a matrix of valid combinations of *fromcode* and *tocode*. The **–l** option is not intended for shell script usage; conforming applications will have to use charmaps.

FUTURE DIRECTIONS

None.

SEE ALSO

gencat

CHANGE HISTORY

First released in Issue 3.

Issue 6

This utility has been rewritten to align with the IEEE P1003.2b draft standard. Specifically, the ability to use charmap files for conversion has been added.

19380 **NAME**

19381 id — return user identity

19382 **SYNOPSIS**

19383 `id [`*`user`*`]`

19384 `id –G[–n] [`*`user`*`]`

19385 `id –g[–nr] [`*`user`*`]`

19386 `id –u[–nr] [`*`user`*`]`

19387 **DESCRIPTION**

19388 If no *user* operand is provided, the *id* utility shall write the user and group IDs and the
19389 corresponding user and group names of the invoking process to standard output. If the effective
19390 and real IDs do not match, both shall be written. If multiple groups are supported by the
19391 underlying system (see the description of {NGROUPS_MAX} in the System Interfaces volume of
19392 IEEE Std 1003.1-200x), the supplementary group affiliations of the invoking process shall also be
19393 written.

19394 If a *user* operand is provided and the process has the appropriate privileges, the user and group
19395 IDs of the selected user shall be written. In this case, effective IDs shall be assumed to be
19396 identical to real IDs. If the selected user has more than one allowable group membership listed
19397 in the group database, these shall be written in the same manner as the supplementary groups
19398 described in the preceding paragraph.

19399 **OPTIONS**

19400 The *id* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
19401 Utility Syntax Guidelines.

19402 The following options shall be supported:

19403 **–G** Output all different group IDs (effective, real, and supplementary) only, using the
19404 format `"%u\n"`. If there is more than one distinct group affiliation, output each
19405 such affiliation, using the format `" %u"`, before the <newline> is output.

19406 **–g** Output only the effective group ID, using the format `"%u\n"`.

19407 **–n** Output the name in the format `"%s"` instead of the numeric ID using the format
19408 `"%u"`.

19409 **–r** Output the real ID instead of the effective ID.

19410 **–u** Output only the effective user ID, using the format `"%u\n"`.

19411 **OPERANDS**

19412 The following operand shall be supported:

19413 *user* The login name for which information is to be written.

19414 **STDIN**

19415 Not used.

19416 **INPUT FILES**

19417 None.

19418 **ENVIRONMENT VARIABLES**

19419 The following environment variables shall affect the execution of *id*:

19420 *LANG* Provide a default value for the internationalization variables that are unset or null.
19421 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
19422 Internationalization Variables for the precedence of internationalization variables

used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The following formats shall be used when the *LC_MESSAGES* locale category specifies the POSIX locale. In other locales, the strings *uid*, *gid*, *euid*, *egid*, and *groups* may be replaced with more appropriate strings corresponding to the locale.

```
"uid=%u(%s) gid=%u(%s)\n", <real user ID>, <user-name>,
    <real group ID>, <group-name>
```

If the effective and real user IDs do not match, the following shall be inserted immediately before the '\n' character in the previous format:

```
" euid=%u(%s)"
```

with the following arguments added at the end of the argument list:

```
<effective user ID>, <effective user-name>
```

If the effective and real group IDs do not match, the following shall be inserted directly before the '\n' character in the format string (and after any addition resulting from the effective and real user IDs not matching):

```
" egid=%u(%s)"
```

with the following arguments added at the end of the argument list:

```
<effective group-ID>, <effective group name>
```

If the process has supplementary group affiliations or the selected user is allowed to belong to multiple groups, the first shall be added directly before the <newline> in the format string:

```
" groups=%u(%s)"
```

with the following arguments added at the end of the argument list:

```
<supplementary group ID>, <supplementary group name>
```

and the necessary number of the following added after that for any remaining supplementary group IDs:

```
",%u(%s)"
```

and the necessary number of the following arguments added at the end of the argument list:

```
<supplementary group ID>, <supplementary group name>
```

If any of the user ID, group ID, effective user ID, effective group ID, or supplementary/multiple group IDs cannot be mapped by the system into printable user or group names, the corresponding ("%s") and *name* argument shall be omitted from the corresponding format string.

When any of the options are specified, the output format shall be as described in the OPTIONS section.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Output produced by the **–G** option and by the default case could potentially produce very long lines on systems that support large numbers of supplementary groups. (On systems with user and group IDs that are 32-bit integers and with group names with a maximum of 8 bytes per name, 93 supplementary groups plus distinct effective and real group and user IDs could theoretically overflow the 2048-byte {LINE_MAX} text file line limit on the default output case. It would take about 186 supplementary groups to overflow the 2048-byte barrier using *id* **–G**). This is not expected to be a problem in practice, but in cases where it is a concern, applications should consider using *fold* **–s** before postprocessing the output of *id*.

EXAMPLES

None.

RATIONALE

The functionality provided by the 4 BSD *groups* utility can be simulated using:

```
id –Gn [ user ]
```

The 4 BSD command *groups* was considered, but it was not included because it did not provide the functionality of the *id* utility of the SVID. Also, it was thought that it would be easier to modify *id* to provide the additional functionality necessary to systems with multiple groups than to invent another command.

The options **–u**, **–g**, **–n**, and **–r** were added to ease the use of *id* with shell commands substitution. Without these options it is necessary to use some preprocessor such as *sed* to select the desired piece of information. Since output such as that produced by:

```
id –u –n
```

is frequently wanted, it seemed desirable to add the options.

FUTURE DIRECTIONS

None.

SEE ALSO

fold, *logname*, *who*, the System Interfaces volume of IEEE Std 1003.1-200x, *getgid*( ), *getgroups*( ), *getuid*( )

CHANGE HISTORY

First released in Issue 2.

19511 **NAME**

19512 ipcrm — remove an XSI message queue, semaphore set, or shared memory segment identifier

19513 **SYNOPSIS**

19514 XSI
```
ipcrm [ –q msgid | –Q msgkey | –s semid | –S semkey |
    –m shmid | –M shmkey ] ...
```
19515

19516

19517 **DESCRIPTION**

19518 The *ipcrm* utility shall remove zero or more message queues, semaphore sets, or shared memory
19519 segments. The interprocess communication facilities to be removed are specified by the options.

19520 Only a user with appropriate privilege shall be allowed to remove an interprocess
19521 communication facility that was not created by or owned by the user invoking *ipcrm*.

19522 **OPTIONS**

19523 The *ipcrm* facility supports the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
19524 Utility Syntax Guidelines.

19525 The following options shall be supported:

19526 **–q** *msgid* Remove the message queue identifier *msgid* from the system and destroy the
19527 message queue and data structure associated with it.

19528 **–m** *shmid* Remove the shared memory identifier *shmid* from the system. The shared memory
19529 segment and data structure associated with it shall be destroyed after the last
19530 detach.

19531 **–s** *semid* Remove the semaphore identifier *semid* from the system and destroy the set of
19532 semaphores and data structure associated with it.

19533 **–Q** *msgkey* Remove the message queue identifier, created with key *msgkey*, from the system
19534 and destroy the message queue and data structure associated with it.

19535 **–M** *shmkey* Remove the shared memory identifier, created with key *shmkey*, from the system.
19536 The shared memory segment and data structure associated with it shall be
19537 destroyed after the last detach.

19538 **–S** *semkey* Remove the semaphore identifier, created with key *semkey*, from the system and
19539 destroy the set of semaphores and data structure associated with it.

19540 **OPERANDS**

19541 None.

19542 **STDIN**

19543 Not used.

19544 **INPUT FILES**

19545 None.

19546 **ENVIRONMENT VARIABLES**

19547 The following environment variables shall affect the execution of *ipcrm*:

19548 *LANG* Provide a default value for the internationalization variables that are unset or null.
19549 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
19550 Internationalization Variables for the precedence of internationalization variables
19551 used to determine the values of locale categories.)

19552 *LC_ALL* If set to a non-empty string value, override the values of all the other
19553 internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH
Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
Not used.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
None.

EXAMPLES
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
ipcs, the System Interfaces volume of IEEE Std 1003.1-200x, *msgctl*( ), *semctl*( ), *shmctl*( )

CHANGE HISTORY
First released in Issue 5.

19590 **NAME**

19591 ipcs — report XSI interprocess communication facilities status

19592 **SYNOPSIS**

19593 XSI
```
ipcs [–qms][–a | –bcopt]
```
19594

19595 **DESCRIPTION**

19596 The *ipcs* utility shall write information about active interprocess communication facilities.

19597 Without options, information shall be written in short format for message queues, shared
19598 memory segments, and semaphores sets that are currently active in the system. Otherwise, the
19599 information that is displayed is controlled by the options specified.

19600 **OPTIONS**

19601 The *ipcs* facility supports the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
19602 Utility Syntax Guidelines.

19603 The *ipcs* utility accepts the following options:

19604 **–q** Write information about active message queues.

19605 **–m** Write information about active shared memory segments.

19606 **–s** Write information about active semaphores sets.

19607 If **–q**, **–m**, or **–s** are specified, only information about those facilities shall be written. If none of
19608 these three are specified, information about all three shall be written subject to the following
19609 options:

19610 **–a** Use all print options. (This is a shorthand notation for **–b**, **–c**, **–o**, **–p**, and **–t**.)

19611 **–b** Write information on maximum allowable size. (Maximum number of bytes in
19612 messages on queue for message queues, size of segments for shared memory, and
19613 number of semaphores in each set for semaphores.)

19614 **–c** Write creator's user name and group name; see below.

19615 **–o** Write information on outstanding usage. (Number of messages on queue and total
19616 number of bytes in messages on queue for message queues, and number of
19617 processes attached to shared memory segments.)

19618 **–p** Write process number information. (Process ID of last process to send a message
19619 and process ID of last process to receive a message on message queues, process ID
19620 of creating process, and process ID of last process to attach or detach on shared
19621 memory segments.)

19622 **–t** Write time information. (Time of the last control operation that changed the access
19623 permissions for all facilities, time of last *msgsnd*( ) and *msgrcv*( ) operations on
19624 message queues, time of last *shmat*( ) and *shmdt*( ) operations on shared memory,
19625 and time of last *semop*( ) operation on semaphores.)

19626 **OPERANDS**

19627 None.

19628 **STDIN**

19629 Not used.

19630 **INPUT FILES**

19631 • The group database

19632 • The user database

19633 **ENVIRONMENT VARIABLES**

19634 The following environment variables shall affect the execution of *ipcs*:

19635 *LANG* Provide a default value for the internationalization variables that are unset or null.
19636 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
19637 Internationalization Variables for the precedence of internationalization variables
19638 used to determine the values of locale categories.)

19639 *LC_ALL* If set to a non-empty string value, override the values of all the other
19640 internationalization variables.

19641 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
19642 characters (for example, single-byte as opposed to multi-byte characters in
19643 arguments).

19644 *LC_MESSAGES*
19645 Determine the locale that should be used to affect the format and contents of
19646 diagnostic messages written to standard error.

19647 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

19648 *TZ* Determine the timezone for the date and time strings written by *ipcs*. If *TZ* is unset
19649 or null, an unspecified default timezone shall be used.

19650 **ASYNCHRONOUS EVENTS**

19651 Default.

19652 **STDOUT**

19653 An introductory line shall be written with the format:

19654 `"IPC status from %s as of %s\n", <source>, <date>`

19655 where <*source*> indicates the source used to gather the statistics and <*date*> is the information
19656 that would be produced by the *date* command when invoked in the POSIX locale.

19657 The *ipcs* utility then shall create up to three reports depending upon the **–q**, **–m**, and **–s** options.
19658 The first report shall indicate the status of message queues, the second report shall indicate the
19659 status of shared memory segments, and the third report shall indicate the status of semaphore
19660 sets.

19661 If the corresponding facility is not installed or has not been used since the last reboot, then the
19662 report shall be written out in the format:

19663 `"%s facility not in system.\n", <facility>`

19664 where <*facility*> is *Message Queue*, *Shared Memory*, or *Semaphore*, as appropriate. If the facility has
19665 been installed and has been used since the last reboot, column headings separated by one or
19666 more spaces and followed by a <newline> shall be written as indicated below followed by the
19667 facility name written out using the format:

19668 `"%s:\n", <facility>`

19669 where <*facility*> is *Message Queues*, *Shared Memory*, or *Semaphores*, as appropriate. On the second
19670 and third reports the column headings need not be written if the last column headings written
19671 already provide column headings for all information in that report.

The column headings provided in the first column below and the meaning of the information in those columns shall be given in order below; the letters in parentheses indicate the options that shall cause the corresponding column to appear; ''all'' means that the column shall always appear. Each column is separated by one or more <space>s. Note that these options only determine what information is provided for each report; they do not determine which reports are written.

T (all) Type of facility:

`q` Message queue.

`m` Shared memory segment.

`s` Semaphore.

This field is a single character written using the format `%c`.

ID (all) The identifier for the facility entry. This field shall be written using the format `%d`.

KEY (all) The key used as an argument to *msgget*( ), *semget*( ), or *shmget*( ) to create the facility entry.

Note: The key of a shared memory segment is changed to IPC_PRIVATE when the segment has been removed until all processes attached to the segment detach it.

This field shall be written using the format `0x%x`.

MODE (all) The facility access modes and flags. The mode shall consist of 11 characters that are interpreted as follows.

The first character shall be:

`S` If a process is waiting on a *msgsnd*( ) operation.

– If the above is not true.

The second character shall be:

`R` If a process is waiting on a *msgrcv*( ) operation.

`C` or – If the associated shared memory segment is to be cleared when the first attach operation is executed.

– If none of the above is true.

The next nine characters shall be interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the usergroup of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is a minus sign (`'–'`).

The permissions shall be indicated as follows:

r If read permission is granted.

w If write permission is granted.

a If alter permission is granted.

– If the indicated permission is not granted.

The first character following the permissions specifies if there is an alternate or additional access control method associated with the facility. If there is no alternate or additional access control method associated with the facility, a single <space> shall be written; otherwise, another printable character is written.

OWNER (all) The user name of the owner of the facility entry. If the user name of the owner is found in the user database, at least the first eight column positions of the name shall be written using the format `%s`. Otherwise, the user ID of the owner shall be written using the format `%d`.

GROUP (all) The group name of the owner of the facility entry. If the group name of the owner is found in the group database, at least the first eight column positions of the name shall be written using the format `%s`. Otherwise, the group ID of the owner shall be written using the format `%d`.

The following nine columns shall be only written out for message queues:

CREATOR (**a**,**c**) The user name of the creator of the facility entry. If the user name of the creator is found in the user database, at least the first eight column positions of the name shall be written using the format `%s`. Otherwise, the user ID of the creator shall be written using the format `%d`.

CGROUP (**a**,**c**) The group name of the creator of the facility entry. If the group name of the creator is found in the group database, at least the first eight column positions of the name shall be written using the format `%s`. Otherwise, the group ID of the creator shall be written using the format `%d`.

CBYTES (**a**,**o**) The number of bytes in messages currently outstanding on the associated message queue. This field shall be written using the format `%d`.

QNUM (**a**,**o**) The number of messages currently outstanding on the associated message queue. This field shall be written using the format `%d`.

QBYTES (**a**,**b**) The maximum number of bytes allowed in messages outstanding on the associated message queue. This field shall be written using the format `%d`.

LSPID (**a**,**p**) The process ID of the last process to send a message to the associated queue. This field shall be written using the format:

`"%d", <`*pid*`>`

where <*pid*> is 0 if no message has been sent to the corresponding message queue; otherwise, <*pid*> shall be the process ID of the last process to send a message to the queue.

LRPID (**a**,**p**) The process ID of the last process to receive a message from the associated queue. This field shall be written using the format:

`"%d", <`*pid*`>`

where <*pid*> is 0 if no message has been received from the corresponding message queue; otherwise, <*pid*> shall be the process ID of the last process to receive a message from the queue.

STIME (**a**,**t**) The time the last message was sent to the associated queue. If a message has been sent to the corresponding message queue, the hour, minute, and second of the last time a message was sent to the queue shall be written using the format `%d:%2.2d:%2.2d`. Otherwise, the format `" no-entry"` shall be written.

RTIME (**a**,**t**) The time the last message was received from the associated queue. If a message has been received from the corresponding message queue, the hour, minute, and second of the last time a message was received from the queue shall be written using the format `%d:%2.2d:%2.2d`. Otherwise, the format `" no-entry"` shall be written.

The following eight columns shall be only written out for shared memory segments.

CREATOR (**a**,**c**) The user of the creator of the facility entry. If the user name of the creator is found in the user database, at least the first eight column positions of the name shall be written using the format `%s`. Otherwise, the user ID of the creator shall be written using the format `%d`.

CGROUP (**a**,**c**) The group name of the creator of the facility entry. If the group name of the creator is found in the group database, at least the first eight column positions of the name shall be written using the format `%s`. Otherwise, the group ID of the creator shall be written using the format `%d`.

NATTCH (**a**,**o**) The number of processes attached to the associated shared memory segment. This field shall be written using the format `%d`.

SEGSZ (**a**,**b**) The size of the associated shared memory segment. This field shall be written using the format `%d`.

CPID (**a**,**p**) The process ID of the creator of the shared memory entry. This field shall be written using the format `%d`.

LPID (**a**,**p**) The process ID of the last process to attach or detach the shared memory segment. This field shall be written using the format:

`"%d", <`*pid*`>`

where <*pid*> is 0 if no process has attached the corresponding shared memory segment; otherwise, <*pid*> shall be the process ID of the last process to attach or detach the segment.

ATIME (**a**,**t**) The time the last attach on the associated shared memory segment was completed. If the corresponding shared memory segment has ever been attached, the hour, minute, and second of the last time the segment was attached shall be written using the format `%d:%2.2d:%2.2d`. Otherwise, the format `" no-entry"` shall be written.

DTIME (**a**,**t**) The time the last detach on the associated shared memory segment was completed. If the corresponding shared memory segment has ever been detached, the hour, minute, and second of the last time the segment was detached shall be written using the format `%d:%2.2d:%2.2d`. Otherwise, the format `" no-entry"` shall be written.

The following four columns shall be only written out for semaphore sets:

CREATOR (**a**,**c**) The user of the creator of the facility entry. If the user name of the creator is found in the user database, at least the first eight column positions of the name shall be written using the format `%s`. Otherwise, the user ID of the creator shall be written using the format `%d`.

CGROUP (**a**,**c**) The group name of the creator of the facility entry. If the group name of the creator is found in the group database, at least the first eight column positions of the name shall be written using the format `%s`. Otherwise, the group ID of the creator shall be written using the format `%d`.

NSEMS (**a**,**b**) The number of semaphores in the set associated with the semaphore entry. This field shall be written using the format %d.

OTIME (**a**,**t**) The time the last semaphore operation on the set associated with the semaphore entry was completed. If a semaphore operation has ever been performed on the corresponding semaphore set, the hour, minute, and second of the last semaphore operation on the semaphore set shall be written using the format %d:%2.2d:%2.2d. Otherwise, the format " no-entry" shall be written.

The following column shall be written for all three reports when it is requested:

CTIME (**a**,**t**) The time the associated entry was created or changed. The hour, minute, and second of the time when the associated entry was created shall be written using the format %d:%2.2d:%2.2d.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Things can change while *ipcs* is running; the information it gives is guaranteed to be accurate only when it was retrieved.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *msgop*( ), *msgrcv*( ), *msgsnd*( ), *semget*( ), *semop*( ), *shmat*( ), *shmdt*( ), *shmget*( ), *shmop*( )

CHANGE HISTORY

First released in Issue 5.

Issue 6

The Open Group Corrigendum U020/1 is applied, correcting the SYNOPSIS.

The Open Group Corrigendum U032/1 and U032/2 are applied, clarifying the output format.

The Open Group Base Resolution bwg98-004 is applied.

19844 **NAME**

19845 jobs — display status of jobs in the current session

19846 **SYNOPSIS**

19847 UP
```
jobs [–l| –p][job_id...]
```
19848

19849 **DESCRIPTION**

19850 The *jobs* utility shall display the status of jobs that were started in the current shell environment;
19851 see Section 2.12 (on page 2263).

19852 When *jobs* reports the termination status of a job, the shell shall remove its process ID from the
19853 list of those ‘‘known in the current shell execution environment’’; see Section 2.9.3.1 (on page
19854 2252).

19855 **OPTIONS**

19856 The *jobs* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
19857 12.2, Utility Syntax Guidelines.

19858 The following options shall be supported:

19859 **–l** (The letter ell.) Provide more information about each job listed. This information
19860 shall include the job number, current job, process group ID, state, and the
19861 command that formed the job.

19862 **–p** Display only the process IDs for the process group leaders of the selected jobs.

19863 By default, the *jobs* utility shall display the status of all stopped jobs, running background jobs
19864 and all jobs whose status has changed and have not been reported by the shell.

19865 **OPERANDS**

19866 The following operand shall be supported:

19867 *job_id* Specifies the jobs for which the status is to be displayed. If no *job_id* is given, the
19868 status information for all jobs shall be displayed. The format of *job_id* is described
19869 in the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.203, Job Control
19870 Job ID.

19871 **STDIN**

19872 Not used.

19873 **INPUT FILES**

19874 None.

19875 **ENVIRONMENT VARIABLES**

19876 The following environment variables shall affect the execution of *jobs*:

19877 *LANG* Provide a default value for the internationalization variables that are unset or null.
19878 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
19879 Internationalization Variables for the precedence of internationalization variables
19880 used to determine the values of locale categories.)

19881 *LC_ALL* If set to a non-empty string value, override the values of all the other
19882 internationalization variables.

19883 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
19884 characters (for example, single-byte as opposed to multi-byte characters in
19885 arguments).

19886 *LC_MESSAGES*
19887 Determine the locale that should be used to affect the format and contents of

diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

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 also shall 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 non-zero exit status, *code*, expressed as a decimal number.

Stopped Indicates that the job was suspended by the SIGTSTP signal.

Stopped (**SIGTSTP**)
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.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The **–p** option is the only portable 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 does not work as expected when it is operating in its own utility execution environment because that environment has no applicable jobs to manipulate. See the APPLICATION USAGE section for *bg* (on page 2410). For this reason, *jobs* is generally implemented as a shell regular built-in.

EXAMPLES

None.

RATIONALE

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 job control features provided by *bg*, *fg*, and *jobs* are based on the KornShell. The standard developers 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 volume of IEEE Std 1003.1-200x to maintain a degree of uniformity with the rest of the KornShell features selected (such as the very popular command line editing features).

The *jobs* utility is not dependent on the job control option, as are the seemingly related *bg* and *fg* utilities because *jobs* is useful for examining background jobs, regardless of the condition of job control. When the user has invoked a *set* **+m** command and job control has been turned off, *jobs* can still be used to examine the background jobs associated with that current session. Similarly, *kill* can then be used to kill background jobs with *kill*% <*background job number*>.

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 understand these formats, although it means that applications 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 early proposal, 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.

FUTURE DIRECTIONS

None.

SEE ALSO

bg, *fg*, *kill*, *wait*

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The JC shading is removed as job control is mandatory in this issue.

19992 **NAME**

19993 join — relational database operator

19994 **SYNOPSIS**

19995 `join [–a` *`file_number`* `| –v` *`file_number`*`][–e` *`string`*`][–o` *`list`*`][–t` *`char`*`]`
19996 `[–1` *`field`*`][–2` *`field`*`]` *`file1 file2`*

19997 **DESCRIPTION**

19998 The *join* utility shall perform an equality join on the files *file1* and *file2*. The joined files shall be
19999 written to the standard output.

20000 The join field is a field in each file on which the files are compared. The *join* utility shall write |
20001 one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The |
20002 output line by default shall consist of the join field, then the remaining fields from *file1*, then the
20003 remaining fields from *file2*. This format can be changed by using the **–o** option (see below). The
20004 **–a** option can be used to add unmatched lines to the output. The **–v** option can be used to output
20005 only unmatched lines.

20006 The files *file1* and *file2* shall be ordered in the collating sequence of *sort* **–b** on the fields on which |
20007 they shall be joined, by default the first in each line. All selected output shall be written in the
20008 same collating sequence.

20009 The default input field separators shall be <blank>s. In this case, multiple separators shall count
20010 as one field separator, and leading separators shall be ignored. The default output field separator
20011 shall be a <space>.

20012 The field separator and collating sequence can be changed by using the **–t** option (see below).

20013 If the same key appears more than once in either file, all combinations of the set of remaining
20014 fields in *file1* and the set of remaining fields in *file2* are output in the order of the lines
20015 encountered.

20016 If the input files are not in the appropriate collating sequence, the results are unspecified.

20017 **OPTIONS**

20018 The *join* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
20019 12.2, Utility Syntax Guidelines.

20020 The following options shall be supported:

20021 **–a** *file_number*

20022 Produce a line for each unpairable line in file *file_number*, where *file_number* is 1 or
20023 2, in addition to the default output. If both **–a**1 and **–a**2 are specified, all unpairable
20024 lines shall be output.

20025 **–e** *string* Replace empty output fields in the list selected by **–o** with the string *string*.

20026 **–o** *list* Construct the output line to comprise the fields specified in *list*, each element of
20027 which shall have one of the following two forms:

20028 1. *file_number.field*, where *file_number* is a file number and *field* is a decimal
20029 integer field number

20030 2. 0 (zero), representing the join field

20031 The elements of *list* shall be either comma-separated or <blank>-separated, as
20032 specified in Guideline 8 of the Base Definitions volume of IEEE Std 1003.1-200x,
20033 Section 12.2, Utility Syntax Guidelines. The fields specified by *list* shall be written
20034 for all selected output lines. Fields selected by *list* that do not appear in the input
20035 shall be treated as empty output fields. (See the **–e** option.) Only specifically

requested fields shall be written. The application shall ensure that *list* is a single command line argument.

–t *char* Use character *char* as a separator, for both input and output. Every appearance of *char* in a line shall be significant. When this option is specified, the collating sequence shall be the same as *sort* without the **–b** option.

–v *file_number*
Instead of the default output, produce a line only for each unpairable line in *file_number*, where *file_number* is 1 or 2. If both **–v**1 and **–v**2 are specified, all unpairable lines shall be output.

–1 *field* Join on the *field*th field of file 1. Fields are decimal integers starting with 1.

–2 *field* Join on the *field*th field of file 2. Fields are decimal integers starting with 1.

OPERANDS

The following operands shall be supported:

file1, *file2*
A pathname of a file to be joined. If either of the *file1* or *file2* operands is '–', the standard input shall be used in its place.

STDIN

The standard input shall be used only if the *file1* or *file2* operand is '–'. See the INPUT FILES section.

INPUT FILES

The input files shall be text files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *join*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale of the collating sequence *join* expects to have been used when the input files were sorted.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

20077 **STDOUT**

20078 The *join* utility output shall be a concatenation of selected character fields. When the **–o** option
20079 is not specified, the output shall be:

20080 `"%s%s%s\n", <join field>, <other file1 fields>,`
20081 `<other file2 fields>`

20082 If the join field is not the first field in a file, the <*other file fields*> for that file shall be:

20083 `<fields preceding join field>, <fields following join field>`

20084 When the **–o** option is specified, the output format shall be:

20085 `"%s\n", <concatenation of fields>`

20086 where the concatenation of fields is described by the **–o** option, above.

20087 For either format, each field (except the last) shall be written with its trailing separator character.
20088 If the separator is the default (<blank>s), a single <space> shall be written after each field
20089 (except the last).

20090 **STDERR**

20091 The standard error shall be used only for diagnostic messages.

20092 **OUTPUT FILES**

20093 None.

20094 **EXTENDED DESCRIPTION**

20095 None.

20096 **EXIT STATUS**

20097 The following exit values shall be returned:

20098 0 All input files were output successfully.

20099 >0 An error occurred.

20100 **CONSEQUENCES OF ERRORS**

20101 Default.

20102 **APPLICATION USAGE**

20103 Pathnames consisting of numeric digits or of the form *string.string* should not be specified
20104 directly following the **–o** list.

20105 **EXAMPLES**

20106 The **–o** 0 field essentially selects the union of the join fields. For example, given file **phone**:

```
20107 !Name           Phone Number
20108 Don             +1 123-456-7890
20109 Hal             +1 234-567-8901
20110 Yasushi         +2 345-678-9012
```

20111 and file **fax**:

```
20112 !Name           Fax Number
20113 Don             +1 123-456-7899
20114 Keith           +1 456-789-0122
20115 Yasushi         +2 345-678-9011
```

20116 (where the large expanses of white space are meant to each represent a single <tab>), the
20117 command:

```
join –t "<tab>" –a 1 –a 2 –e '(unknown)' –o 0,1.2,2.2 phone fax
```

would produce:

```
!Name           Phone Number            Fax Number
Don             +1 123-456-7890         +1 123-456-7899
Hal             +1 234-567-8901         (unknown)
Keith           (unknown)               +1 456-789-0122
Yasushi         +2 345-678-9012         +2 345-678-9011
```

Multiple instances of the same key will produce combinatorial results. The following:

```
fa:
    a x
    a y
    a z
fb:
    a p
```

will produce:

```
a x p
a y p
a z p
```

And the following:

```
fa:
    a b c
    a d e
fb:
    a w x
    a y z
    a o p
```

will produce:

```
a b c w x
a b c y z
a b c o p
a d e w x
a d e y z
a d e o p
```

RATIONALE

The **–e** option is only effective when used with **–o** because, unless specific fields are identified using **–o**, *join* is not aware of what fields might be empty. The exception to this is the join field, but identifying an empty join field with the **–e** string is not historical practice and some scripts might break if this were changed.

The 0 field in the **–o** list was adopted from the Tenth Edition version of *join* to satisfy international objections that the *join* in the base documents do not support the ''full join'' or ''outer join'' described in relational database literature. Although it has been possible to include a join field in the output (by default, or by field number using **–o**), the join field could not be included for an unpaired line selected by **–a**. The **–o** 0 field essentially selects the union of the join fields.

This sort of outer join was not possible with the *join* commands in the base documents. The **–o** 0 field was chosen because it is an upward-compatible change for applications. An alternative was

considered: have the join field represent the union of the fields in the files (where they are identical for matched lines, and one or both are null for unmatched lines). This was not adopted because it would break some historical applications.

The ability to specify *file2* as – is not historical practice; it was added for completeness.

The **–v** option is not historical practice, but was considered necessary because it permitted the writing of *only* those lines that do not match on the join field, as opposed to the **–a** option, which prints both lines that do and do not match. This additional facility is parallel with the **–v** option of *grep*.

Some historical implementations have been encountered where a blank line in one of the input files was considered to be the end of the file; the description in this volume of IEEE Std 1003.1-200x does not cite this as an allowable case.

FUTURE DIRECTIONS

None.

SEE ALSO

awk, *comm*, *sort*, *uniq*

CHANGE HISTORY

First released in Issue 2.

Issue 6

The obsolescent **–j** options and the multi-argument **–o** option are withdrawn in this issue.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

NAME

kill — terminate or signal processes

SYNOPSIS

```
kill –s signal_name pid ...
kill –l [exit_status]
```

XSI

```
kill [–signal_name] pid ...
kill [–signal_number] pid ...
```

DESCRIPTION

The *kill* utility shall send a signal to the process or processes specified by each *pid* operand.

For each *pid* operand, the *kill* utility shall perform actions equivalent to the *kill*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x called with the following arguments:

- The value of the *pid* operand shall be used as the *pid* argument.
- The *sig* argument is the value specified by the **–s** option, *–signal_number* option, or the *–signal_name* option, or by SIGTERM, if none of these options is specified.

OPTIONS

The *kill* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines, except that in the last two SYMOPSIS forms, the *–signal_number* and *–signal_name* options are usually more than a single character.

The following options shall be supported:

–l (The letter ell.) Write all values of *signal_name* supported by the implementation, if no operand is given. If an *exit_status* operand is given and it is a value of the `'?'` shell special parameter (see Section 2.5.2 (on page 2235) and *wait* (on page 3239)) corresponding to a process that was terminated by a signal, the *signal_name* corresponding to the signal that terminated the process shall be written. If an *exit_status* operand is given and it is the unsigned decimal integer value of a signal number, the *signal_name* (the symbolic constant name without the **SIG** prefix defined in the Base Definitions volume of IEEE Std 1003.1-200x) corresponding to that signal shall be written. Otherwise, the results are unspecified.

–s *signal_name*
Specify the signal to send, using one of the symbolic names defined in the **<signal.h>** header. Values of *signal_name* shall be recognized in a case-independent fashion, without the **SIG** prefix. In addition, the symbolic name 0 shall be recognized, representing the signal value zero. The corresponding signal shall be sent instead of SIGTERM.

XSI

–signal_name
Equivalent to **–s** *signal_name*.

XSI

–signal_number
Specify a non-negative decimal integer, *signal_number*, representing the signal to be used instead of SIGTERM, as the *sig* argument in the effective call to *kill*( ). The correspondence between integer values and the *sig* value used is shown in the following table.

The effects of specifying any *signal_number* other than those listed in the table are undefined.

signal_number	*sig* **Value**
0	0
1	SIGHUP
2	SIGINT
3	SIGQUIT
6	SIGABRT
9	SIGKILL
14	SIGALRM
15	SIGTERM

If the first argument is a negative integer, it shall be interpreted as a *–signal_number* option, not as a negative *pid* operand specifying a process group.

OPERANDS

The following operands shall be supported:

pid One of the following:

1. A decimal integer specifying a process or process group to be signaled. The process or processes selected by positive, negative and zero values of the *pid* operand shall be as described for the *kill*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x. If process number 0 is specified, all processes in the current process group shall be signaled. For the effects of negative *pid* numbers, see the *kill*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x. If the first *pid* operand is negative, it should be preceded by "−−" to keep it from being interpreted as an option.
2. A job control job ID (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.203, Job Control Job ID) 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 Section 2.12 (on page 2263).

exit_status A decimal integer specifying a signal number or the exit status of a process terminated by a signal.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *kill*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
When the **–l** option is not specified, the standard output shall not be used.

When the **–l** option is specified, the symbolic name of each signal shall be written in the following format:

```
"%s%c", <signal_name>, <separator>
```

where the <*signal_name*> is in uppercase, without the **SIG** prefix, and the <*separator*> shall be either a <newline> or a <space>. For the last signal written, <*separator*> shall be a <newline>.

When both the **–l** option and *exit_status* operand are specified, the symbolic name of the corresponding signal shall be written in the following format:

```
"%s\n", <signal_name>
```

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 At least one matching process was found for each *pid* operand, and the specified signal was successfully processed for at least one matching process.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
Process numbers can be found by using *ps*.

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 does not share the shell's understanding of job numbers.

EXAMPLES
Any of the commands:

```
kill –9 100 –165
kill –s kill 100 –165
kill –s KILL 100 –165
```

sends the SIGKILL signal to the process whose process ID is 100 and to all processes whose process group ID is 165, assuming the sending process has permission to send that signal to the specified processes, and that they exist.

The System Interfaces volume of IEEE Std 1003.1-200x and this volume of IEEE Std 1003.1-200x do not require specific signal numbers for any *signal_names*. Even the *–signal_number* option provides symbolic (although numeric) names for signals. If a process is terminated by a signal, its exit status indicates the signal that killed it, but the exact values are not specified. The *kill* **–l** option, however, can be used to map decimal signal numbers and exit status values into the name of a signal. The following example reports the status of a terminated job:

```
job
stat=$?
if [ $stat –eq 0 ]
then
    echo job completed successfully.
elif [ $stat –gt 128 ]
then
    echo job terminated by signal SIG$(kill –l $stat).
else
    echo job terminated with error code $stat.
fi
```

To send the default signal to a process group (say 123), an application should use a command similar to one of the following:

```
kill –TERM –123
kill –– –123
```

RATIONALE

The **–l** option originated from the C shell, and is also implemented in the KornShell. The C shell output can consist of multiple output lines because the signal names do not always fit on a single line on some terminal screens. The KornShell output also included the implementation-defined signal numbers and was considered by the standard developers to be too difficult for scripts to parse conveniently. The specified output format is intended not only to accommodate the historical C shell output, but also to permit an entirely vertical or entirely horizontal listing on systems for which this is appropriate.

An early proposal invented the name SIGNULL as a *signal_name* for signal 0 (used by the System Interfaces volume of IEEE Std 1003.1-200x to test for the existence of a process without sending it a signal). Since the *signal_name* 0 can be used in this case unambiguously, SIGNULL has been removed.

An early proposal also required symbolic *signal_name*s to be recognized with or without the **SIG** prefix. Historical versions of *kill* have not written the **SIG** prefix for the **–l** option and have not recognized the **SIG** prefix on *signal_name*s. Since neither applications portability nor ease-of-use would be improved by requiring this extension, it is no longer required.

To avoid an ambiguity of an initial negative number argument specifying either a signal number or a process group, IEEE Std 1003.1-200x mandates that it is always considered the former by implementations that support the XSI option. It also requires that conforming applications always use the "––" options terminator argument when specifying a process group, unless an option is also specified.

The **–s** option was added in response to international interest in providing some form of *kill* that meets the Utility Syntax Guidelines.

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 does not understand how the shell has managed its job numbers.

FUTURE DIRECTIONS

None.

SEE ALSO

ps, *wait*, the System Interfaces volume of IEEE Std 1003.1-200x, *kill*( ), the Base Definitions volume of IEEE Std 1003.1-200x, **<signal.h>**

CHANGE HISTORY

First released in Issue 2.

Issue 6

The obsolescent versions of the SYNOPSIS were turned into non-obsolescent features of the XSI option, corresponding to a similar change in the *trap* special built-in.

20379 **NAME**

20380 lex — generate programs for lexical tasks (**DEVELOPMENT**)

20381 **SYNOPSIS**

20382 CD
```
lex [–t][–n|–v][file ...]
```
20383

20384 **DESCRIPTION**

20385 The *lex* utility shall generate C programs to be used in lexical processing of character input, and
20386 that can be used as an interface to *yacc*. The C programs shall be generated from *lex* source code
20387 and conform to the ISO C standard. Usually, the *lex* utility shall write the program it generates to
20388 the file **lex.yy.c**; the state of this file is unspecified if *lex* exits with a non-zero exit status. See the
20389 EXTENDED DESCRIPTION section for a complete description of the *lex* input language.

20390 **OPTIONS**

20391 The *lex* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
20392 Utility Syntax Guidelines.

20393 The following options shall be supported:

20394 **–n** Suppress the summary of statistics usually written with the **–v** option. If no table
20395 sizes are specified in the *lex* source code and the **–v** option is not specified, then **–n**
20396 is implied.

20397 **–t** Write the resulting program to standard output instead of **lex.yy.c**.

20398 **–v** Write a summary of *lex* statistics to the standard output. (See the discussion of *lex*
20399 table sizes in **Definitions in lex** (on page 2736).) If the **–t** option is specified and
20400 **–n** is not specified, this report shall be written to standard error. If table sizes are
20401 specified in the *lex* source code, and if the **–n** option is not specified, the **–v** option
20402 may be enabled.

20403 **OPERANDS**

20404 The following operand shall be supported:

20405 *file* A pathname of an input file. If more than one such *file* is specified, all files shall be
20406 concatenated to produce a single *lex* program. If no *file* operands are specified, or if
20407 a *file* operand is '–', the standard input shall be used.

20408 **STDIN**

20409 The standard input shall be used if no *file* operands are specified, or if a *file* operand is '–'. See
20410 INPUT FILES.

20411 **INPUT FILES**

20412 The input files shall be text files containing *lex* source code, as described in the EXTENDED
20413 DESCRIPTION section.

20414 **ENVIRONMENT VARIABLES**

20415 The following environment variables shall affect the execution of *lex*:

20416 *LANG* Provide a default value for the internationalization variables that are unset or null.
20417 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
20418 Internationalization Variables for the precedence of internationalization variables
20419 used to determine the values of locale categories.)

20420 *LC_ALL* If set to a non-empty string value, override the values of all the other
20421 internationalization variables.

20422 *LC_COLLATE*
20423 Determine the locale for the behavior of ranges, equivalence classes and multi-

20424 character collating elements within regular expressions. If this variable is not set to
20425 the POSIX locale, the results are unspecified.

20426 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
20427 characters (for example, single-byte as opposed to multi-byte characters in
20428 arguments and input files), and the behavior of character classes within regular
20429 expressions. If this variable is not set to the POSIX locale, the results are
20430 unspecified.

20431 *LC_MESSAGES*
20432 Determine the locale that should be used to affect the format and contents of
20433 diagnostic messages written to standard error.

20434 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

20435 **ASYNCHRONOUS EVENTS**
20436 Default.

20437 **STDOUT**
20438 If the **–t** option is specified, the text file of C source code output of *lex* shall be written to
20439 standard output.

20440 If the **–t** option is not specified:

20441 • Implementation-defined informational, error, and warning messages concerning the contents
20442 of *lex* source code input shall be written to either the standard output or standard error.

20443 • If the **–v** option is specified and the **–n** option is not specified, *lex* statistics shall also be
20444 written to either the standard output or standard error, in an implementation-defined format.
20445 These statistics may also be generated if table sizes are specified with a '%' operator in the
20446 *Definitions* section, as long as the **–n** option is not specified.

20447 **STDERR**
20448 If the **–t** option is specified, implementation-defined informational, error, and warning messages
20449 concerning the contents of *lex* source code input shall be written to the standard error.

20450 If the **–t** option is not specified:

20451 1. Implementation-defined informational, error, and warning messages concerning the
20452 contents of *lex* source code input shall be written to either the standard output or standard
20453 error.

20454 2. If the **–v** option is specified and the **–n** option is not specified, *lex* statistics shall also be
20455 written to either the standard output or standard error, in an implementation-defined
20456 format. These statistics may also be generated if table sizes are specified with a '%'
20457 operator in the *Definitions* section, as long as the **–n** option is not specified.

20458 **OUTPUT FILES**
20459 A text file containing C source code shall be written to **lex.yy.c**, or to the standard output if the
20460 **–t** option is present.

20461 **EXTENDED DESCRIPTION**
20462 Each input file shall contain *lex* source code, which is a table of regular expressions with |
20463 corresponding actions in the form of C program fragments.

20464 When **lex.yy.c** is compiled and linked with the *lex* library (using the **–l l** operand with *c99*), the |
20465 resulting program shall read character input from the standard input and shall partition it into |
20466 strings that match the given expressions. |

When an expression is matched, these actions shall occur:

- The input string that was matched shall be left in *yytext* as a null-terminated string; *yytext* shall either be an external character array or a pointer to a character string. As explained in **Definitions in lex**, the type can be explicitly selected using the **%array** or **%pointer** declarations, but the default is implementation-defined.
- The external **int** *yyleng* shall be set to the length of the matching string.
- The expression's corresponding program fragment, or action, shall be executed.

During pattern matching, *lex* shall search the set of patterns for the single longest possible match. Among rules that match the same number of characters, the rule given first shall be chosen.

The general format of *lex* source shall be:

Definitions
%%
Rules
%%
*User*Subroutines

The first "%%" is required to mark the beginning of the rules (regular expressions and actions); the second "%%" is required only if user subroutines follow.

Any line in the *Definitions* section beginning with a <blank> shall be assumed to be a C program fragment and shall be copied to the external definition area of the **lex.yy.c** file. Similarly, anything in the *Definitions* section included between delimiter lines containing only "%{" and "%}" shall also be copied unchanged to the external definition area of the **lex.yy.c** file.

Any such input (beginning with a <blank> or within "%{" and "%}" delimiter lines) appearing at the beginning of the *Rules* section before any rules are specified shall be written to **lex.yy.c** after the declarations of variables for the *yylex*() function and before the first line of code in *yylex*(). Thus, user variables local to *yylex*() can be declared here, as well as application code to execute upon entry to *yylex*().

The action taken by *lex* when encountering any input beginning with a <blank> or within "%{" and "%}" delimiter lines appearing in the *Rules* section but coming after one or more rules is undefined. The presence of such input may result in an erroneous definition of the *yylex*() function.

Definitions in lex

Definitions appear before the first "%%" delimiter. Any line in this section not contained between "%{" and "%}" lines and not beginning with a <blank> shall be assumed to define a *lex* substitution string. The format of these lines shall be:

name substitute

If a *name* does not meet the requirements for identifiers in the ISO C standard, the result is undefined. The string *substitute* shall replace the string {*name*} when it is used in a rule. The *name* string shall be recognized in this context only when the braces are provided and when it does not appear within a bracket expression or within double-quotes.

In the *Definitions* section, any line beginning with a '%' (percent sign) character and followed by an alphanumeric word beginning with either 's' or 'S' shall define a set of start conditions. Any line beginning with a '%' followed by a word beginning with either 'x' or 'X' shall define a set of exclusive start conditions. When the generated scanner is in a "%s" state, patterns with

no state specified shall be also active; in a "%x" state, such patterns shall not be active. The rest of the line, after the first word, shall be considered to be one or more <blank>-separated names of start conditions. Start condition names shall be constructed in the same way as definition names. Start conditions can be used to restrict the matching of regular expressions to one or more states as described in **Regular Expressions in lex** (on page 2738).

Implementations shall accept either of the following two mutually exclusive declarations in the *Definitions* section:

%array Declare the type of *yytext* to be a null-terminated character array.

%pointer Declare the type of *yytext* to be a pointer to a null-terminated character string.

The default type of *yytext* is implementation-defined. If an application refers to *yytext* outside of the scanner source file (that is, via an **extern**), the application shall include the appropriate **%array** or **%pointer** declaration in the scanner source file.

Implementations shall accept declarations in the *Definitions* section for setting certain internal table sizes. The declarations are shown in the following table.

Table 4-9 Table Size Declarations in *lex*

Declaration	Description	Minimum Value
%**p** *n*	Number of positions	2 500
%**n** *n*	Number of states	500
%**a** *n*	Number of transitions	2 000
%**e** *n*	Number of parse tree nodes	1 000
%**k** *n*	Number of packed character classes	1 000
%**o** *n*	Size of the output array	3 000

In the table, *n* represents a positive decimal integer, preceded by one or more <blank>s. The exact meaning of these table size numbers is implementation-defined. The implementation shall document how these numbers affect the *lex* utility and how they are related to any output that may be generated by the implementation should limitations be encountered during the execution of *lex*. It shall be possible to determine from this output which of the table size values needs to be modified to permit *lex* to successfully generate tables for the input language. The values in the column Minimum Value represent the lowest values conforming implementations shall provide.

Rules in lex

The rules in *lex* source files are a table in which the left column contains regular expressions and the right column contains actions (C program fragments) to be executed when the expressions are recognized.

```
ERE action
ERE action
...
```

The extended regular expression (*ERE*) portion of a row shall be separated from *action* by one or more <blank>s. A regular expression containing <blank>s shall be recognized under one of the following conditions:

- The entire expression appears within double-quotes.
- The <blank>s appear within double-quotes or square brackets.

- Each <blank> is preceded by a backslash character.

User Subroutines in lex

Anything in the user subroutines section shall be copied to **lex.yy.c** following *yylex*( ).

Regular Expressions in lex

The *lex* utility shall support the set of extended regular expressions (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.4, Extended Regular Expressions), with the following additions and exceptions to the syntax:

`"..."` Any string enclosed in double-quotes shall represent the characters within the double-quotes as themselves, except that backslash escapes (which appear in the following table) shall be recognized. Any backslash-escape sequence shall be terminated by the closing quote. For example, `"\01""1"` represents a single string: the octal value 1 followed by the character `'1'`.

<state>r, *<state1,state2,...>r*
The regular expression *r* shall be matched only when the program is in one of the start conditions indicated by *state*, *state1*, and so on; see **Actions in lex** (on page 2740). (As an exception to the typographical conventions of the rest of this volume of IEEE Std 1003.1-200x, in this case *<state>* does not represent a metavariable, but the literal angle-bracket characters surrounding a symbol.) The start condition shall be recognized as such only at the beginning of a regular expression.

r/*x* The regular expression *r* shall be matched only if it is followed by an occurrence of regular expression *x* (*x* is the instance of trailing context, further defined below). The token returned in *yytext* shall only match *r*. If the trailing portion of *r* matches the beginning of *x*, the result is unspecified. The *r* expression cannot include further trailing context or the `'$'` (match-end-of-line) operator; *x* cannot include the `'^'` (match-beginning-of-line) operator, nor trailing context, nor the `'$'` operator. That is, only one occurrence of trailing context is allowed in a *lex* regular expression, and the `'^'` operator only can be used at the beginning of such an expression.

{*name*} When *name* is one of the substitution symbols from the *Definitions* section, the string, including the enclosing braces, shall be replaced by the *substitute* value. The *substitute* value shall be treated in the extended regular expression as if it were enclosed in parentheses. No substitution shall occur if {*name*} occurs within a bracket expression or within double-quotes.

Within an ERE, a backslash character shall be considered to begin an escape sequence as specified in the table in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`). In addition, the escape sequences in the following table shall be recognized.

A literal <newline> cannot occur within an ERE; the escape sequence `'\n'` can be used to represent a <newline>. A <newline> shall not be matched by a period operator.

20592 **Table 4-10** Escape Sequences in *lex*

20593 20594 Escape Sequence	Description	Meaning
20595 20596 20597 20598 20599 20600 20601 20602 20603 20604 20605 20606 `\`*digits*	A backslash character followed by the longest sequence of one, two, or three octal-digit characters (01234567). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the one, two, or three-digit octal integer. If the size of a byte on the system is greater than nine bits, the valid escape sequence used to represent a byte is implementation-defined. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading `'\'` for each byte.
20607 20608 20609 20610 20611 20612 20613 20614 `\x`*digits*	A backslash character followed by the longest sequence of hexadecimal-digit characters (01234567abcdefABCDEF). If all of the digits are 0 (that is, representation of the NUL character), the behavior is undefined.	The character whose encoding is represented by the hexadecimal integer.
20615 20616 20617 20618 20619 20620 20621 20622 `\c`	A backslash character followed by any character not described in this table or in the table in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`).	The character `'c'`, unchanged.

20623 **Note:** If a `'\x'` sequence needs to be immediately followed by a hexadecimal digit character, a
20624 sequence such as `"\x1""1"` can be used, which represents a character containing the value 1,
20625 followed by the character `'1'`.

20626 The order of precedence given to extended regular expressions for *lex* differs from that specified
20627 in the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.4, Extended Regular
20628 Expressions. The order of precedence for *lex* shall be as shown in the following table, from high
20629 to low.

20630 **Note:** The escaped characters entry is not meant to imply that these are operators, but they are
20631 included in the table to show their relationships to the true operators. The start condition,
20632 trailing context, and anchoring notations have been omitted from the table because of the
20633 placement restrictions described in this section; they can only appear at the beginning or
20634 ending of an ERE.

20635 **Table 4-11** ERE Precedence in *lex*

Extended Regular Expression	Precedence
collation-related bracket symbols	`[= =]  [: :]  [. .]`
escaped characters	`\<special character>`
bracket expression	`[ ]`
quoting	`"..."`
grouping	`( )`
definition	`{name}`
single-character RE duplication	`* + ?`
concatenation	
interval expression	`{m,n}`
alternation	`|`

20647 The ERE anchoring operators '^' and '$') do not appear in the table. With *lex* regular
20648 expressions, these operators are restricted in their use: the '^' operator can only be used at the
20649 beginning of an entire regular expression, and the '$' operator only at the end. The operators
20650 apply to the entire regular expression. Thus, for example, the pattern "(^abc)|(def$)" is
20651 undefined; it can instead be written as two separate rules, one with the regular expression
20652 "^abc" and one with "def$", which share a common action via the special '|' action (see
20653 below). If the pattern were written "^abc|def$", it would match either "abc" or "def" on a
20654 line by itself.

20655 Unlike the general ERE rules, embedded anchoring is not allowed by most historical *lex*
20656 implementations. An example of embedded anchoring would be for patterns such as
20657 "(^|)foo(|$)" to match "foo" when it exists as a complete word. This functionality can
20658 be obtained using existing *lex* features:

```
20659 ^foo/[ \n]       |
20660 " foo"/[ \n]     /* Found foo as a separate word. */
```

20661 Note also that '$' is a form of trailing context (it is equivalent to "/\n") and as such cannot be
20662 used with regular expressions containing another instance of the operator (see the preceding
20663 discussion of trailing context).

20664 The additional regular expressions trailing-context operator '/' can be used as an ordinary
20665 character if presented within double-quotes, "/"; preceded by a backslash, "\/"; or within a
20666 bracket expression, "[/]". The start-condition '<' and '>' operators shall be special only in a
20667 start condition at the beginning of a regular expression; elsewhere in the regular expression they
20668 shall be treated as ordinary characters.

20669 **Actions in lex**

20670 The action to be taken when an ERE is matched can be a C program fragment or the special
20671 actions described below; the program fragment can contain one or more C statements, and can
20672 also include special actions. The empty C statement ';' shall be a valid action; any string in the
20673 **lex.yy.c** input that matches the pattern portion of such a rule is effectively ignored or skipped.
20674 However, the absence of an action shall not be valid, and the action *lex* takes in such a condition
20675 is undefined.

20676 The specification for an action, including C statements and special actions, can extend across
20677 several lines if enclosed in braces:

```
20678 ERE <one or more blanks> { program statement
20679                            program statement }
```

The default action when a string in the input to a **lex.yy.c** program is not matched by any expression shall be to copy the string to the output. Because the default behavior of a program generated by *lex* is to read the input and copy it to the output, a minimal *lex* source program that has just "%%" shall generate a C program that simply copies the input to the output unchanged.

Four special actions shall be available:

```
|    ECHO;    REJECT;    BEGIN
```

| The action '|' means that the action for the next rule is the action for this rule. Unlike the other three actions, '|' cannot be enclosed in braces or be semicolon-terminated; the application shall ensure that it is specified alone, with no other actions.

ECHO; Write the contents of the string *yytext* on the output.

REJECT; Usually only a single expression is matched by a given string in the input. **REJECT** means ''continue to the next expression that matches the current input'', and shall cause whatever rule was the second choice after the current rule to be executed for the same input. Thus, multiple rules can be matched and executed for one input string or overlapping input strings. For example, given the regular expressions "xyz" and "xy" and the input "xyz", usually only the regular expression "xyz" would match. The next attempted match would start after **z.** If the last action in the "xyz" rule is **REJECT**, both this rule and the "xy" rule would be executed. The **REJECT** action may be implemented in such a fashion that flow of control does not continue after it, as if it were equivalent to a **goto** to another part of *yylex*(). The use of **REJECT** may result in somewhat larger and slower scanners.

BEGIN The action:

```
BEGIN newstate;
```

switches the state (start condition) to *newstate*. If the string *newstate* has not been declared previously as a start condition in the *Definitions* section, the results are unspecified. The initial state is indicated by the digit '0' or the token **INITIAL**.

The functions or macros described below are accessible to user code included in the *lex* input. It is unspecified whether they appear in the C code output of *lex*, or are accessible only through the **–l l** operand to *c99* (the *lex* library).

int *yylex*(**void**)
Performs lexical analysis on the input; this is the primary function generated by the *lex* utility. The function shall return zero when the end of input is reached; otherwise, it shall return non-zero values (tokens) determined by the actions that are selected.

int *yymore*(**void**)
When called, indicates that when the next input string is recognized, it is to be appended to the current value of *yytext* rather than replacing it; the value in *yyleng* shall be adjusted accordingly.

int *yyless*(**int** *n*)
Retains *n* initial characters in *yytext*, NUL-terminated, and treats the remaining characters as if they had not been read; the value in *yyleng* shall be adjusted accordingly.

int *input*(**void**)
Returns the next character from the input, or zero on end-of-file. It shall obtain input from the stream pointer *yyin*, although possibly via an intermediate buffer. Thus, once scanning has begun, the effect of altering the value of *yyin* is undefined. The character read shall be removed from the input stream of the scanner without any processing by the scanner.

int *unput*(**int** *c*)
Returns the character `'c'` to the input; *yytext* and *yyleng* are undefined until the next expression is matched. The result of using *unput*( ) for more characters than have been input is unspecified.

The following functions shall appear only in the *lex* library accessible through the **–l l** operand; they can therefore be redefined by a conforming application:

int *yywrap*(**void**)
Called by *yylex*( ) at end-of-file; the default *yywrap*( ) shall always return 1. If the application requires *yylex*( ) to continue processing with another source of input, then the application can include a function *yywrap*( ), which associates another file with the external variable **FILE *** *yyin* and shall return a value of zero.

int *main*(**int** *argc*, **char ***argv*[ ])
Calls *yylex*( ) to perform lexical analysis, then exits. The user code can contain *main*( ) to perform application-specific operations, calling *yylex*( ) as applicable.

Except for *input*( ), *unput*( ), and *main*( ), all external and static names generated by *lex* shall begin with the prefix **yy** or **YY**.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Conforming applications are warned that in the *Rules* section, an *ERE* without an action is not acceptable, but need not be detected as erroneous by *lex*. This may result in compilation or runtime errors.

The purpose of *input*( ) is to take characters off the input stream and discard them as far as the lexical analysis is concerned. A common use is to discard the body of a comment once the beginning of a comment is recognized.

The *lex* utility is not fully internationalized in its treatment of regular expressions in the *lex* source code or generated lexical analyzer. It would seem desirable to have the lexical analyzer interpret the regular expressions given in the *lex* source according to the environment specified when the lexical analyzer is executed, but this is not possible with the current *lex* technology. Furthermore, the very nature of the lexical analyzers produced by *lex* must be closely tied to the lexical requirements of the input language being described, which is frequently locale-specific anyway. (For example, writing an analyzer that is used for French text is not automatically useful for processing other languages.)

EXAMPLES

The following is an example of a *lex* program that implements a rudimentary scanner for a Pascal-like syntax:

```
%{
/* Need this for the call to atof() below. */
#include <math.h>
/* Need this for printf(), fopen(), and stdin below. */
#include <stdio.h>
%}
```

```
DIGIT    [0–9]
ID       [a–z][a–z0–9]*

%%

{DIGIT}+ {
    printf("An integer: %s (%d)\n", yytext,
        atoi(yytext));
    }

{DIGIT}+"."{DIGIT}*        {
    printf("A float: %s (%g)\n", yytext,
        atof(yytext));
    }

if|then|begin|end|procedure|function        {
    printf("A keyword: %s\n", yytext);
    }

{ID}    printf("An identifier: %s\n", yytext);

"+"|"–"|"*"|"/"        printf("An operator: %s\n", yytext);

"{"[^}\n]*"}"    /* Eat up one-line comments. */

[ \t\n]+        /* Eat up white space. */

.  printf("Unrecognized character: %s\n", yytext);

%%

int main(int argc, char *argv[])
{
    ++argv, --argc;  /* Skip over program name. */
    if (argc > 0)
        yyin = fopen(argv[0], "r");
    else
        yyin = stdin;

    yylex();
}
```

RATIONALE

Even though the **–c** option and references to the C language are retained in this description, *lex* may be generalized to other languages, as was done at one time for EFL, the Extended FORTRAN Language. Since the *lex* input specification is essentially language-independent, versions of this utility could be written to produce Ada, Modula-2, or Pascal code, and there are known historical implementations that do so.

The current description of *lex* bypasses the issue of dealing with internationalized EREs in the *lex* source code or generated lexical analyzer. If it follows the model used by *awk* (the source code is assumed to be presented in the POSIX locale, but input and output are in the locale specified by the environment variables), then the tables in the lexical analyzer produced by *lex* would interpret EREs specified in the *lex* source in terms of the environment variables specified when *lex* was executed. The desired effect would be to have the lexical analyzer interpret the EREs given in the *lex* source according to the environment specified when the lexical analyzer is executed, but this is not possible with the current *lex* technology.

The description of octal and hexadecimal-digit escape sequences agrees with the ISO C standard use of escape sequences. See the RATIONALE for *ed* (on page 2537) for a discussion of bytes

larger than 9 bits being represented by octal values. Hexadecimal values can represent larger bytes and multi-byte characters directly, using as many digits as required.

There is no detailed output format specification. The observed behavior of *lex* under four different historical implementations was that none of these implementations consistently reported the line numbers for error and warning messages. Furthermore, there was a desire that *lex* be allowed to output additional diagnostic messages. Leaving message formats unspecified avoids these formatting questions and problems with internationalization.

Although the %x specifier for *exclusive* start conditions is not historical practice, it is believed to be a minor change to historical implementations and greatly enhances the usability of *lex* programs since it permits an application to obtain the expected functionality with fewer statements.

The **%array** and **%pointer** declarations were added as a compromise between historical systems. The System V-based *lex* copies the matched text to a *yytext* array. The *flex* program, supported in BSD and GNU systems, uses a pointer. In the latter case, significant performance improvements are available for some scanners. Most historical programs should require no change in porting from one system to another because the string being referenced is null-terminated in both cases. (The method used by *flex* in its case is to null-terminate the token in place by remembering the character that used to come right after the token and replacing it before continuing on to the next scan.) Multi-file programs with external references to *yytext* outside the scanner source file should continue to operate on their historical systems, but would require one of the new declarations to be considered strictly portable.

The description of EREs avoids unnecessary duplication of ERE details because their meanings within a *lex* ERE are the same as that for the ERE in this volume of IEEE Std 1003.1-200x.

The reason for the undefined condition associated with text beginning with a <blank> or within "%{" and "%}" delimiter lines appearing in the *Rules* section is historical practice. Both the BSD and System V *lex* copy the indented (or enclosed) input in the *Rules* section (except at the beginning) to unreachable areas of the *yylex*( ) function (the code is written directly after a *break* statement). In some cases, the System V *lex* generates an error message or a syntax error, depending on the form of indented input.

The intention in breaking the list of functions into those that may appear in **lex.yy.c** *versus* those that only appear in **libl.a** is that only those functions in **libl.a** can be reliably redefined by a conforming application.

The descriptions of standard output and standard error are somewhat complicated because historical *lex* implementations chose to issue diagnostic messages to standard output (unless **–t** was given). This standard allows this behavior, but leaves an opening for the more expected behavior of using standard error for diagnostics. Also, the System V behavior of writing the statistics when any table sizes are given is allowed, while BSD-derived systems can avoid it. The programmer can always precisely obtain the desired results by using either the **–t** or **–n** options.

The OPERANDS section does not mention the use of – as a synonym for standard input; not all historical implementations support such usage for any of the *file* operands.

A description of the *translation table* was deleted from early proposals because of its relatively low usage in historical applications.

The change to the definition of the *input*( ) function that allows buffering of input presents the opportunity for major performance gains in some applications.

The following examples clarify the differences between *lex* regular expressions and regular expressions appearing elsewhere in this volume of IEEE Std 1003.1-200x. For regular expressions of the form "r/x", the string matching *r* is always returned; confusion may arise when the

beginning of *x* matches the trailing portion of *r*. For example, given the regular expression `"a*b/cc"` and the input `"aaabcc"`, *yytext* would contain the string `"aaab"` on this match. But given the regular expression `"x*/xy"` and the input `"xxxy"`, the token **xxx**, not **xx**, is returned by some implementations because **xxx** matches `"x*"`.

In the rule `"ab*/bc"`, the `"b*"` at the end of *r* extends *r*'s match into the beginning of the trailing context, so the result is unspecified. If this rule were `"ab/bc"`, however, the rule matches the text `"ab"` when it is followed by the text `"bc"`. In this latter case, the matching of *r* cannot extend into the beginning of *x*, so the result is specified.

FUTURE DIRECTIONS

None.

SEE ALSO

c99, *yacc*

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is now marked as part of the C-Language Development Utilities option.

The obsolescent **–c** option is withdrawn in this issue.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

20882 **NAME**

20883 link — call *link*( ) function

20884 **SYNOPSIS**

20885 XSI
```
link file1 file2
```
20886

20887 **DESCRIPTION**

20888 The *link* utility shall perform the function call:

20889
```
link(file1, file2);
```

20890 A user may need appropriate privilege to invoke the *link* utility.

20891 **OPTIONS**

20892 None.

20893 **OPERANDS**

20894 The following operands shall be supported:

20895 *file1* The pathname of an existing file.

20896 *file2* The pathname of the new directory entry to be created.

20897 **STDIN**

20898 Not used.

20899 **INPUT FILES**

20900 Not used.

20901 **ENVIRONMENT VARIABLES**

20902 The following environment variables shall affect the execution of *link*:

20903 *LANG* Provide a default value for the internationalization variables that are unset or null.
20904 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
20905 Internationalization Variables for the precedence of internationalization variables
20906 used to determine the values of locale categories.)

20907 *LC_ALL* If set to a non-empty string value, override the values of all the other
20908 internationalization variables.

20909 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
20910 characters (for example, single-byte as opposed to multi-byte characters in
20911 arguments).

20912 *LC_MESSAGES*
20913 Determine the locale that should be used to affect the format and contents of
20914 diagnostic messages written to standard error.

20915 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

20916 **ASYNCHRONOUS EVENTS**

20917 Default.

20918 **STDOUT**

20919 None.

20920 **STDERR**

20921 The standard error shall be used only for diagnostic messages. |

20922 **OUTPUT FILES**

20923 None.

20924 **EXTENDED DESCRIPTION**

20925 None.

20926 **EXIT STATUS**

20927 The following exit values shall be returned:

20928 0 Successful completion.

20929 >0 An error occurred.

20930 **CONSEQUENCES OF ERRORS**

20931 Default.

20932 **APPLICATION USAGE**

20933 None.

20934 **EXAMPLES**

20935 None.

20936 **RATIONALE**

20937 None.

20938 **FUTURE DIRECTIONS**

20939 None.

20940 **SEE ALSO**

20941 *ln*, *unlink*, the System Interfaces volume of IEEE Std 1003.1-200x, *link*( )

20942 **CHANGE HISTORY**

20943 First released in Issue 5.

20944 **NAME**

20945 ln — link files

20946 **SYNOPSIS**

20947 `ln [–fs] source_file target_file`

20948 `ln [–fs] source_file ... target_dir`

20949 **DESCRIPTION**

20950 In the first synopsis form, the *ln* utility shall create a new directory entry (link) at the destination
20951 path specified by the *target_file* operand. If the **–s** option is specified, a symbolic link shall be
20952 created for the file specified by the *source_file* operand. This first synopsis form shall be assumed
20953 when the final operand does not name an existing directory; if more than two operands are
20954 specified and the final is not an existing directory, an error shall result.

20955 In the second synopsis form, the *ln* utility shall create a new directory entry (link), or if the **–s**
20956 option is specified a symbolic link, for each file specified by a *source_file* operand, at a *destination*
20957 path in the existing directory named by *target_dir*.

20958 If the last operand specifies an existing file of a type not specified by the System Interfaces
20959 volume of IEEE Std 1003.1-200x, the behavior is implementation-defined.

20960 The corresponding *destination* path for each *source_file* shall be the concatenation of the target
20961 directory pathname, a slash character, and the last pathname component of the *source_file*. The
20962 second synopsis form shall be assumed when the final operand names an existing directory.

20963 For each *source_file*:

20964 1. If the *destination* path exists:

20965 a. If the **–f** option is not specified, *ln* shall write a diagnostic message to standard error,
20966 do nothing more with the current *source_file*, and go on to any remaining *source_files*.

20967 b. Actions shall be performed equivalent to the *unlink*() function defined in the System
20968 Interfaces volume of IEEE Std 1003.1-200x, called using *destination* as the *path*
20969 argument. If this fails for any reason, *ln* shall write a diagnostic message to standard
20970 error, do nothing more with the current *source_file*, and go on to any remaining
20971 *source_files*.

20972 2. If the **–s** option is specified, *ln* shall create a symbolic link named by the *destination* path
20973 and containing as its pathname *source_file*. The *ln* utility shall do nothing more with
20974 *source_file* and shall go on to any remaining files.

20975 3. If *source_file* is a symbolic link, actions shall be performed equivalent to the *link*() function
20976 using the object that *source_file* references as the *path1* argument and the destination path
20977 as the *path2* argument. The *ln* utility shall do nothing more with *source_file* and shall go on
20978 to any remaining files.

20979 4. Actions shall be performed equivalent to the *link*() function defined in the System
20980 Interfaces volume of IEEE Std 1003.1-200x using *source_file* as the *path1* argument, and the
20981 *destination* path as the *path2* argument.

20982 **OPTIONS**

20983 The *ln* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
20984 Utility Syntax Guidelines.

20985 The following option shall be supported:

20986 **–f** Force existing *destination* pathnames to be removed to allow the link.

20987 **–s** Create symbolic links instead of hard links.

20988 **OPERANDS**

20989 The following operands shall be supported:

20990 *source_file* A pathname of a file to be linked. If the **–s** option is specified, no restrictions on the
20991 type of file or on its existence shall be made. If the **–s** option is not specified,
20992 whether a directory can be linked is implementation-defined.

20993 *target_file* The pathname of the new directory entry to be created.

20994 *target_dir* A pathname of an existing directory in which the new directory entries are created.

20995 **STDIN**

20996 Not used.

20997 **INPUT FILES**

20998 None.

20999 **ENVIRONMENT VARIABLES**

21000 The following environment variables shall affect the execution of *ln*:

21001 *LANG* Provide a default value for the internationalization variables that are unset or null.
21002 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
21003 Internationalization Variables for the precedence of internationalization variables
21004 used to determine the values of locale categories.)

21005 *LC_ALL* If set to a non-empty string value, override the values of all the other
21006 internationalization variables.

21007 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
21008 characters (for example, single-byte as opposed to multi-byte characters in
21009 arguments).

21010 *LC_MESSAGES*
21011 Determine the locale that should be used to affect the format and contents of
21012 diagnostic messages written to standard error.

21013 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

21014 **ASYNCHRONOUS EVENTS**

21015 Default.

21016 **STDOUT**

21017 Not used.

21018 **STDERR**

21019 The standard error shall be used only for diagnostic messages.

21020 **OUTPUT FILES**

21021 None.

21022 **EXTENDED DESCRIPTION**

21023 None.

21024 **EXIT STATUS**

21025 The following exit values shall be returned:

21026 0 All the specified files were linked successfully.

21027 >0 An error occurred.

21028 **CONSEQUENCES OF ERRORS**
21029 Default.

21030 **APPLICATION USAGE**
21031 None.

21032 **EXAMPLES**
21033 None.

21034 **RATIONALE**
21035 Some historic versions of *ln* (including the one specified by the SVID, unlink the destination file,
21036 if it exists, by default. If the mode does not permit writing, these versions prompt for
21037 confirmation before attempting the unlink. In these versions the **–f** option causes *ln* not to
21038 attempt to prompt for confirmation.

21039 This allows *ln* to succeed in creating links when the target file already exists, even if the file itself
21040 is not writable (although the directory must be). Early proposals specified this functionality.

21041 This volume of IEEE Std 1003.1-200x does not allow the *ln* utility to unlink existing destination
21042 paths by default for the following reasons:

21043 • The *ln* utility has historically been used to provide locking for shell applications, a usage that
21044 is incompatible with *ln* unlinking the destination path by default. There was no
21045 corresponding technical advantage to adding this functionality.

21046 • This functionality gave *ln* the ability to destroy the link structure of files, which changes the
21047 historical behavior of *ln*.

21048 • This functionality is easily replicated with a combination of *rm* and *ln*.

21049 • It is not historical practice in many systems; BSD and BSD-derived systems do not support
21050 this behavior. Unfortunately, whichever behavior is selected can cause scripts written
21051 expecting the other behavior to fail.

21052 • It is preferable that *ln* perform in the same manner as the *link*( ) function, which does not
21053 permit the target to exist already.

21054 This volume of IEEE Std 1003.1-200x retains the **–f** option to provide support for shell scripts
21055 depending on the SVID semantics. It seems likely that shell scripts would not be written to
21056 handle prompting by *ln* and would therefore have specified the **–f** option.

21057 The **–f** option is an undocumented feature of many historical versions of the *ln* utility, allowing
21058 linking to directories. These versions require modification.

21059 Early proposals of this volume of IEEE Std 1003.1-200x also required an **–i** option, which
21060 behaved like the **–i** options in *cp* and *mv*, prompting for confirmation before unlinking existing
21061 files. This was not historical practice for the *ln* utility and has been omitted.

21062 **FUTURE DIRECTIONS**
21063 None.

21064 **SEE ALSO**
21065 *chmod*, *find*, *pax*, *rm*, the System Interfaces volume of IEEE Std 1003.1-200x, *link*( )

21066 **CHANGE HISTORY**
21067 First released in Issue 2.

Issue 6

The *ln* utility is updated to include symbolic link processing as defined in the IEEE P1003.2b draft standard.

NAME

locale — get locale-specific information

SYNOPSIS

```
locale [–a| –m]
locale [–ck] name...
```

DESCRIPTION

The *locale* utility shall write information about the current locale environment, or all public locales, to the standard output. For the purposes of this section, a *public locale* is one provided by the implementation that is accessible to the application.

When *locale* is invoked without any arguments, it shall summarize the current locale environment for each locale category as determined by the settings of the environment variables defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale.

When invoked with operands, it shall write values that have been assigned to the keywords in the locale categories, as follows:

- Specifying a keyword name shall select the named keyword and the category containing that keyword.
- Specifying a category name shall select the named category and all keywords in that category.

OPTIONS

The *locale* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–a Write information about all available public locales. The available locales shall include **POSIX**, representing the POSIX locale. The manner in which the implementation determines what other locales are available is implementation-defined.

–c Write the names of selected locale categories; see the STDOUT section. The **–c** option increases readability when more than one category is selected (for example, via more than one keyword name or via a category name). It is valid both with and without the **–k** option.

–k Write the names and values of selected keywords. The implementation may omit values for some keywords; see the OPERANDS section.

–m Write names of available charmaps; see the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set.

OPERANDS

The following operand shall be supported:

name The name of a locale category as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale, the name of a keyword in a locale category, or the reserved name **charmap**. The named category or keyword shall be selected for output. If a single *name* represents both a locale category name and a keyword name in the current locale, the results are unspecified. Otherwise, both category and keyword names can be specified as *name* operands, in any sequence. It is implementation-defined whether any keyword values are written for the categories *LC_CTYPE* and *LC_COLLATE*.

21115 **STDIN**

21116 Not used.

21117 **INPUT FILES**

21118 None.

21119 **ENVIRONMENT VARIABLES**

21120 The following environment variables shall affect the execution of *locale*:

21121 *LANG* Provide a default value for the internationalization variables that are unset or null.
21122 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
21123 Internationalization Variables for the precedence of internationalization variables
21124 used to determine the values of locale categories.)

21125 *LC_ALL* If set to a non-empty string value, override the values of all the other
21126 internationalization variables.

21127 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
21128 characters (for example, single-byte as opposed to multi-byte characters in
21129 arguments and input files).

21130 *LC_MESSAGES*
21131 Determine the locale that should be used to affect the format and contents of
21132 diagnostic messages written to standard error.

21133 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

21134 XSI The application shall ensure that the *LANG*, *LC_**, and *NLSPATH* environment variables specify
21135 the current locale environment to be written out; they shall be used if the **–a** option is not
21136 specified.

21137 **ASYNCHRONOUS EVENTS**

21138 Default.

21139 **STDOUT**

21140 If *locale* is invoked without any options or operands, the names and values of the *LANG* and
21141 *LC_** environment variables described in this volume of IEEE Std 1003.1-200x shall be written to
21142 the standard output, one variable per line, with *LANG* first, and each line using the following
21143 format. Only those variables set in the environment and not overridden by *LC_ALL* shall be
21144 written using this format:

21145 `"%s=%s\n", <variable_name>, <value>`

21146 The names of those *LC_** variables associated with locale categories defined in this volume of
21147 IEEE Std 1003.1-200x that are not set in the environment or are overridden by *LC_ALL* shall be
21148 written in the following format:

21149 `"%s=\""%s\""\n", <variable_name>, <implied value>`

21150 The <*implied value*> shall be the name of the locale that has been selected for that category by the
21151 implementation, based on the values in *LANG* and *LC_ALL*, as described in the Base Definitions
21152 volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

21153 The <*value*> and <*implied value*> shown above shall be properly quoted for possible later reentry
21154 to the shell. The <*value*> shall not be quoted using double-quotes (so that it can be distinguished
21155 by the user from the <*implied value*> case, which always requires double-quotes).

21156 The *LC_ALL* variable shall be written last, using the first format shown above. If it is not set, it
21157 shall be written as:

21158 `"LC_ALL=\n"`

21159 If any arguments are specified:

21160 1. If the **–a** option is specified, the names of all the public locales shall be written, each in the
21161 following format:

21162 `"%s\n", <locale name>`

21163 2. If the **–c** option is specified, the names of all selected categories shall be written, each in the
21164 following format:

21165 `"%s\n", <category name>`

21166 If keywords are also selected for writing (see following items), the category name output
21167 shall precede the keyword output for that category.

21168 If the **–c** option is not specified, the names of the categories shall not be written; only the
21169 keywords, as selected by the <*name*> operand, shall be written.

21170 3. If the **–k** option is specified, the names and values of selected keywords shall be written. If
21171 a value is non-numeric, it shall be written in the following format:

21172 `"%s=\"%s\"\n", <keyword name>, <keyword value>`

21173 If the keyword was **charmap**, the name of the charmap (if any) that was specified via the
21174 *localedef* **–f** option when the locale was created shall be written, with the word **charmap** as
21175 <*keyword name*>.

21176 If a value is numeric, it shall be written in one of the following formats:

21177 `"%s=%d\n", <keyword name>, <keyword value>`

21178 `"%s=%c%o\n", <keyword name>, <escape character>, <keyword value>`

21179 `"%s=%cx%x\n", <keyword name>, <escape character>, <keyword value>`

21180 where the <*escape character*> is that identified by the **escape_char** keyword in the current
21181 locale; see the Base Definitions volume of IEEE Std 1003.1-200x, Section 7.3, Locale
21182 Definition.

21183 Compound keyword values (list entries) shall be separated in the output by semicolons.
21184 When included in keyword values, the semicolon, the double-quote, the backslash, and
21185 any control character shall be preceded (escaped) with the escape character.

21186 4. If the **–k** option is not specified, selected keyword values shall be written, each in the
21187 following format:

21188 `"%s\n", <keyword value>`

21189 If the keyword was **charmap**, the name of the charmap (if any) that was specified via the
21190 *localedef* **–f** option when the locale was created shall be written.

21191 5. If the **–m** option is specified, then a list of all available charmaps shall be written, each in
21192 the format:

21193 `"%s\n", <charmap>`

21194 where <*charmap*> is in a format suitable for use as the option-argument to the *localedef* **–f**
21195 option.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All the requested information was found and output successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

If the *LANG* environment variable is not set or set to an empty value, or one of the *LC_** environment variables is set to an unrecognized value, the actual locales assumed (if any) are implementation-defined as described in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

Implementations are not required to write out the actual values for keywords in the categories *LC_CTYPE* and *LC_COLLATE*; however, they must write out the categories (allowing an application to determine, for example, which character classes are available).

EXAMPLES

In the following examples, the assumption is that locale environment variables are set as follows:

```
LANG=locale_x
LC_COLLATE=locale_y
```

The command *locale* would result in the following output:

```
LANG=locale_x
LC_CTYPE="locale_x"
LC_COLLATE=locale_y
LC_TIME="locale_x"
LC_NUMERIC="locale_x"
LC_MONETARY="locale_x"
LC_MESSAGES="locale_x"
LC_ALL=
```

The order of presentation of the categories is not specified by this volume of IEEE Std 1003.1-200x.

The command:

```
LC_ALL=POSIX locale –ck decimal_point
```

would produce:

```
LC_NUMERIC
decimal_point="."
```

The following command shows an application of *locale* to determine whether a user-supplied response is affirmative:

```
21239       if printf "%s\n" "$response" | grep –Eq "$(locale yesexpr)"
21240       then
21241           affirmative processing goes here
21242       else
21243           non-affirmative processing goes here
21244       fi
```

21245 **RATIONALE**

21246 The output for categories *LC_CTYPE* and *LC_COLLATE* has been made implementation-defined
21247 because there is a questionable value in having a shell script receive an entire array of characters.
21248 It is also difficult to return a logical collation description, short of returning a complete *localedef*
21249 source.

21250 The **–m** option was included to allow applications to query for the existence of charmaps. The
21251 output is a list of the charmaps (implementation-supplied and user-supplied, if any) on the
21252 system.

21253 The **–c** option was included for readability when more than one category is selected (for
21254 example, via more than one keyword name or via a category name). It is valid both with and
21255 without the **–k** option.

21256 The **charmap** keyword, which returns the name of the charmap (if any) that was used when the
21257 current locale was created, was included to allow applications needing the information to
21258 retrieve it.

21259 **FUTURE DIRECTIONS**

21260 None.

21261 **SEE ALSO**

21262 *localedef*, the Base Definitions volume of IEEE Std 1003.1-200x, Section 7.3, Locale Definition

21263 **CHANGE HISTORY**

21264 First released in Issue 4.

21265 **Issue 5**

21266 FUTURE DIRECTIONS section added.

21267 **Issue 6**

21268 The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

NAME

localedef — define locale environment

SYNOPSIS

```
localedef [–c][–f charmap][–i sourcefile][–u code_set_name] name
```

DESCRIPTION

The *localedef* utility shall convert source definitions for locale categories into a format usable by the functions and utilities whose operational behavior is determined by the setting of the locale environment variables defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale. It is implementation-defined whether users have the capability to create new locales, in addition to those supplied by the implementation. If the symbolic constant POSIX2_LOCALEDEF is defined, the system supports the creation of new locales. XSI On XSI-conformant systems, the symbolic constant POSIX2_LOCALEDEF shall be defined.

The utility shall read source definitions for one or more locale categories belonging to the same locale from the file named in the **–i** option (if specified) or from standard input.

The *name* operand identifies the target locale. The utility shall support the creation of *public*, or generally accessible locales, as well as *private*, or restricted-access locales. Implementations may restrict the capability to create or modify public locales to users with the appropriate privileges.

Each category source definition shall be identified by the corresponding environment variable name and terminated by an **END** *category-name* statement. The following categories shall be supported. In addition, the input may contain source for implementation-defined categories.

LC_CTYPE Defines character classification and case conversion.

LC_COLLATE
Defines collation rules.

LC_MONETARY
Defines the format and symbols used in formatting of monetary information.

LC_NUMERIC
Defines the decimal delimiter, grouping, and grouping symbol for non-monetary numeric editing.

LC_TIME Defines the format and content of date and time information.

LC_MESSAGES
Defines the format and values of affirmative and negative responses.

OPTIONS

The *localedef* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–c Create permanent output even if warning messages have been issued.

–f *charmap* Specify the pathname of a file containing a mapping of character symbols and collating element symbols to actual character encodings. The format of the *charmap* is described under the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.4, Character Set Description File. The application shall ensure that this option is specified if symbolic names (other than collating symbols defined in a **collating-symbol** keyword) are used. If the **–f** option is not present, an implementation-defined character mapping shall be used.

21312 **–i** *inputfile* The pathname of a file containing the source definitions. If this option is not
21313 present, source definitions shall be read from standard input. The format of the
21314 *inputfile* is described in the Base Definitions volume of IEEE Std 1003.1-200x,
21315 Section 7.3, Locale Definition.

21316 **–u** *code_set_name* |
21317 Specify the name of a codeset used as the target mapping of character symbols and |
21318 collating element symbols whose encoding values are defined in terms of the
21319 ISO / IEC 10646-1: 2000 standard position constant values.

21320 **OPERANDS**

21321 The following operand shall be supported:

21322 *name* Identifies the locale; see the Base Definitions volume of IEEE Std 1003.1-200x,
21323 Chapter 7, Locale for a description of the use of this name. If the name contains one
21324 or more slash characters, *name* shall be interpreted as a pathname where the
21325 created locale definitions shall be stored. If *name* does not contain any slash
21326 characters, the interpretation of the name is implementation-defined and the locale
21327 shall be public. This capability may be restricted to users with appropriate
21328 privileges. (As a consequence of specifying one *name*, although several categories
21329 can be processed in one execution, only categories belonging to the same locale can
21330 be processed.)

21331 **STDIN**

21332 Unless the **–i** option is specified, the standard input shall be a text file containing one or more
21333 locale category source definitions, as described in the Base Definitions volume of
21334 IEEE Std 1003.1-200x, Section 7.3, Locale Definition. When lines are continued using the escape
21335 character mechanism, there is no limit to the length of the accumulated continued line.

21336 **INPUT FILES**

21337 The character set mapping file specified as the *charmap* option-argument is described under the
21338 Base Definitions volume of IEEE Std 1003.1-200x, Section 6.4, Character Set Description File. If a
21339 locale category source definition contains a **copy** statement, as defined in the Base Definitions
21340 volume of IEEE Std 1003.1-200x, Chapter 7, Locale, and the **copy** statement names a valid,
21341 existing locale, then *localedef* shall behave as if the source definition had contained a valid
21342 category source definition for the named locale.

21343 **ENVIRONMENT VARIABLES**

21344 The following environment variables shall affect the execution of *localedef*:

21345 *LANG* Provide a default value for the internationalization variables that are unset or null.
21346 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
21347 Internationalization Variables for the precedence of internationalization variables
21348 used to determine the values of locale categories.)

21349 *LC_ALL* If set to a non-empty string value, override the values of all the other
21350 internationalization variables.

21351 *LC_COLLATE*
21352 (This variable has no affect on *localedef*; the POSIX locale is used for this category.)

21353 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
21354 characters (for example, single-byte as opposed to multi-byte characters in
21355 arguments and input files). This variable has no affect on the processing of *localedef*
21356 input data; the POSIX locale is used for this purpose, regardless of the value of this
21357 variable.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
The utility shall report all categories successfully processed, in an unspecified format.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
The format of the created output is unspecified. If the *name* operand does not contain a slash, the existence of an output file for the locale is unspecified.

EXTENDED DESCRIPTION
When the **–u** option is used, the *code_set_name* option-argument shall be interpreted as an implementation-defined name of a codeset to which the ISO/IEC 10646-1:2000 standard position constant values shall be converted via an implementation-defined method. Both the ISO/IEC 10646-1:2000 standard position constant values and other formats (decimal, hexadecimal, or octal) shall be valid as encoding values within the *charmap* file. The codeset represented by the implementation-defined name can be any codeset that is supported by the implementation.

When conflicts occur between the *charmap* specification of <*code_set_name*>, <*mb_cur_max*>, or <*mb_cur_min*> and the implementation-defined interpretation of these respective items for the codeset represented by the **–u** option-argument *code_set_name*, the result is unspecified.

When conflicts occur between the *charmap* encoding values specified for symbolic names of characters of the portable character set and the implementation-defined assignment of character encoding values, the result is unspecified.

If a non-printable character in the *charmap* has a width specified that is not **–1**, *localedef* shall generate a warning.

EXIT STATUS
The following exit values shall be returned:

0 No errors occurred and the locales were successfully created.

1 Warnings occurred and the locales were successfully created.

2 The locale specification exceeded implementation limits or the coded character set or sets used were not supported by the implementation, and no locale was created.

3 The capability to create new locales is not supported by the implementation.

>3 Warnings or errors occurred and no output was created.

CONSEQUENCES OF ERRORS
If an error is detected, no permanent output shall be created.

If warnings occur, permanent output shall be created if the **–c** option was specified. The following conditions shall cause warning messages to be issued:

- If a symbolic name not found in the *charmap* file is used for the descriptions of the *LC_CTYPE* or *LC_COLLATE* categories (for other categories, this shall be an error condition).

- If the number of operands to the **order** keyword exceeds the {COLL_WEIGHTS_MAX} limit.
- If optional keywords not supported by the implementation are present in the source.
- If a non-printable character has a width specified other than –1.

Other implementation-defined conditions may also cause warnings.

APPLICATION USAGE

The *charmap* definition is optional, and is contained outside the locale definition. This allows both completely self-defined source files, and generic sources (applicable to more than one codeset). To aid portability, all *charmap* definitions must use the same symbolic names for the portable character set. As explained in the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.4, Character Set Description File, it is implementation-defined whether or not users or applications can provide additional character set description files. Therefore, the **–f** option might be operable only when an implementation-defined *charmap* is named.

EXAMPLES

None.

RATIONALE

The output produced by the *localedef* utility is implementation-defined. The *name* operand is used to identify the specific locale. (As a consequence, although several categories can be processed in one execution, only categories belonging to the same locale can be processed.)

FUTURE DIRECTIONS

None.

SEE ALSO

locale, the Base Definitions volume of IEEE Std 1003.1-200x, Section 7.3, Locale Definition

CHANGE HISTORY

First released in Issue 4.

Issue 6

The **–u** option is added, as specified in the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

21428 **NAME**

21429 logger — log messages

21430 **SYNOPSIS**

21431 `logger` *`string`* `...`

21432 **DESCRIPTION**

21433 The *logger* utility saves a message, in an unspecified manner and format, containing the *string*
21434 operands provided by the user. The messages are expected to be evaluated later by personnel
21435 performing system administration tasks.

21436 It is implementation-defined whether messages written in locales other than the POSIX locale
21437 are effective.

21438 **OPTIONS**

21439 None.

21440 **OPERANDS**

21441 The following operand shall be supported:

21442 *string* One of the string arguments whose contents are concatenated together, in the
21443 order specified, separated by single <space>s.

21444 **STDIN**

21445 Not used.

21446 **INPUT FILES**

21447 None.

21448 **ENVIRONMENT VARIABLES**

21449 The following environment variables shall affect the execution of *logger*:

21450 *LANG* Provide a default value for the internationalization variables that are unset or null.
21451 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
21452 Internationalization Variables for the precedence of internationalization variables
21453 used to determine the values of locale categories.)

21454 *LC_ALL* If set to a non-empty string value, override the values of all the other
21455 internationalization variables.

21456 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
21457 characters (for example, single-byte as opposed to multi-byte characters in
21458 arguments).

21459 *LC_MESSAGES*
21460 Determine the locale that should be used to affect the format and contents of
21461 diagnostic messages written to standard error. (This means diagnostics from *logger*
21462 to the user or application, not diagnostic messages that the user is sending to the
21463 system administrator.)

21464 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

21465 **ASYNCHRONOUS EVENTS**

21466 Default.

21467 **STDOUT**

21468 Not used.

21469 **STDERR**

21470 The standard error shall be used only for diagnostic messages.

21471 **OUTPUT FILES**

21472 Unspecified.

21473 **EXTENDED DESCRIPTION**

21474 None.

21475 **EXIT STATUS**

21476 The following exit values shall be returned:

21477 0 Successful completion.

21478 >0 An error occurred.

21479 **CONSEQUENCES OF ERRORS**

21480 Default.

21481 **APPLICATION USAGE**

21482 This utility allows logging of information for later use by a system administrator or programmer
21483 in determining why non-interactive utilities have failed. The locations of the saved messages,
21484 their format, and retention period are all unspecified. There is no method for a conforming
21485 application to read messages, once written.

21486 **EXAMPLES**

21487 A batch application, running non-interactively, tries to read a configuration file and fails; it may
21488 attempt to notify the system administrator with:

21489
```
logger myname: unable to read file foo. [timestamp]
```

21490 **RATIONALE**

21491 The standard developers believed strongly that some method of alerting administrators to errors
21492 was necessary. The obvious example is a batch utility, running non-interactively, that is unable
21493 to read its configuration files or that is unable to create or write its results file. However, the
21494 standard developers did not wish to define the format or delivery mechanisms as they have
21495 historically been (and will probably continue to be) very system-specific, as well as involving
21496 functionality clearly outside of the scope of this volume of IEEE Std 1003.1-200x.

21497 The text with *LC_MESSAGES* about diagnostic messages means diagnostics from *logger* to the
21498 user or application, not diagnostic messages that the user is sending to the system administrator.

21499 Multiple *string* arguments are allowed, similar to *echo*, for ease-of-use.

21500 Like the utilities *mailx* and *lp*, *logger* is admittedly difficult to test. This was not deemed sufficient
21501 justification to exclude these utilities from this volume of IEEE Std 1003.1-200x. It is also
21502 arguable that they are, in fact, testable, but that the tests themselves are not portable.

21503 **FUTURE DIRECTIONS**

21504 None.

21505 **SEE ALSO**

21506 *mailx*, *write*

21507 **CHANGE HISTORY**

21508 First released in Issue 4.

21509 **NAME**

21510 logname — return the user's login name

21511 **SYNOPSIS**

21512 `logname`

21513 **DESCRIPTION**

21514 The *logname* utility shall write the user's login name to standard output. The login name shall be
21515 the string that would be returned by the *getlogin*( ) function defined in the System Interfaces
21516 volume of IEEE Std 1003.1-200x. Under the conditions where the *getlogin*( ) function would fail,
21517 the *logname* utility shall write a diagnostic message to standard error and exit with a non-zero
21518 exit status.

21519 **OPTIONS**

21520 None.

21521 **OPERANDS**

21522 None.

21523 **STDIN**

21524 Not used.

21525 **INPUT FILES**

21526 None.

21527 **ENVIRONMENT VARIABLES**

21528 The following environment variables shall affect the execution of *logname*:

21529 *LANG* Provide a default value for the internationalization variables that are unset or null.
21530 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
21531 Internationalization Variables for the precedence of internationalization variables
21532 used to determine the values of locale categories.)

21533 *LC_ALL* If set to a non-empty string value, override the values of all the other
21534 internationalization variables.

21535 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
21536 characters (for example, single-byte as opposed to multi-byte characters in
21537 arguments).

21538 *LC_MESSAGES*
21539 Determine the locale that should be used to affect the format and contents of
21540 diagnostic messages written to standard error.

21541 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

21542 **ASYNCHRONOUS EVENTS**

21543 Default.

21544 **STDOUT**

21545 The *logname* utility output shall be a single line consisting of the user's login name:

21546 `"%s\n", <login name>`

21547 **STDERR**

21548 The standard error shall be used only for diagnostic messages.

21549 **OUTPUT FILES**

21550 None.

21551 **EXTENDED DESCRIPTION**

21552 None.

21553 **EXIT STATUS**

21554 The following exit values shall be returned:

21555 0 Successful completion.

21556 >0 An error occurred.

21557 **CONSEQUENCES OF ERRORS**

21558 Default.

21559 **APPLICATION USAGE**

21560 The *logname* utility explicitly ignores the *LOGNAME* environment variable because environment
21561 changes could produce erroneous results.

21562 **EXAMPLES**

21563 None.

21564 **RATIONALE**

21565 The **passwd** file is not listed as required because the implementation may have other means of
21566 mapping login names.

21567 **FUTURE DIRECTIONS**

21568 None.

21569 **SEE ALSO**

21570 *id*, *who*

21571 **CHANGE HISTORY**

21572 First released in Issue 2.

21573 **NAME**

21574 lp — send files to a printer

21575 **SYNOPSIS**

21576 `lp [–c][–d dest][–n copies][–msw][–o option]... [–t title][file...]`

21577 **DESCRIPTION**

21578 The *lp* utility shall copy the input files to an output destination in an unspecified manner. The
21579 default output destination should be to a hardcopy device, such as a printer or microfilm
21580 recorder, that produces non-volatile, human-readable documents. If such a device is not
21581 available to the application, or if the system provides no such device, the *lp* utility shall exit with
21582 a non-zero exit status.

21583 The actual writing to the output device may occur some time after the *lp* utility successfully
21584 exits. During the portion of the writing that corresponds to each input file, the implementation
21585 shall guarantee exclusive access to the device.

21586 The *lp* utility shall associate a unique *request ID* with each request.

21587 Normally, a banner page is produced to separate and identify each print job. This page may be
21588 suppressed by implementation-defined conditions, such as an operator command or one of the
21589 **–o** *option* values.

21590 **OPTIONS**

21591 The *lp* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
21592 Utility Syntax Guidelines.

21593 The following options shall be supported:

21594 **–c** Exit only after further access to any of the input files is no longer required. The
21595 application can then safely delete or modify the files without affecting the output
21596 operation. Normally, files are not copied, but are linked whenever possible. If the
21597 **–c** option is not given, then the user should be careful not to remove any of the
21598 files before the request has been printed in its entirety. It should also be noted that
21599 in the absence of the **–c** option, any changes made to the named files after the
21600 request is made but before it is printed may be reflected in the printed output. On |
21601 some implementations, **–c** may be on by default. |

21602 **–d** *dest* Specify a string that names the destination (*dest*). If *dest* is a printer, the request
21603 shall be printed only on that specific printer. If *dest* is a class of printers, the request
21604 shall be printed on the first available printer that is a member of the class. Under
21605 certain conditions (printer unavailability, file space limitation, and so on), requests
21606 for specific destinations need not be accepted. Destination names vary between
21607 systems.

21608 If **–d** is not specified, and neither the *LPDEST* nor *PRINTER* environment variable
21609 is set, an unspecified destination is used. The **–d** *dest* option shall take precedence
21610 over *LPDEST*, which in turn shall take precedence over *PRINTER*. Results are
21611 undefined when *dest* contains a value that is not a valid destination name.

21612 **–m** Send mail (see *mailx* (on page 2785)) after the files have been printed. By default,
21613 no mail is sent upon normal completion of the print request.

21614 **–n** *copies* Write *copies* number of copies of the files, where *copies* is a positive decimal integer.
21615 The methods for producing multiple copies and for arranging the multiple copies
21616 when multiple *file* operands are used are unspecified, except that each file shall be
21617 output as an integral whole, not interleaved with portions of other files.

21618 **–o** *option* Specify printer-dependent or class-dependent *option*s. Several such *option*s may be
21619 collected by specifying the **–o** option more than once.

21620 **–s** Suppress messages from *lp*.

21621 **–t** *title* Write *title* on the banner page of the output.

21622 **–w** Write a message on the user's terminal after the files have been printed. If the user
21623 is not logged in, then mail shall be sent instead.

21624 **OPERANDS**

21625 The following operand shall be supported:

21626 *file* A pathname of a file to be output. If no *file* operands are specified, or if a *file*
21627 operand is '–', the standard input shall be used. If a *file* operand is used, but the
21628 **–c** option is not specified, the process performing the writing to the output device
21629 may have user and group permissions that differ from that of the process invoking
21630 *lp*.

21631 **STDIN**

21632 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '–'. |
21633 See the INPUT FILES section.

21634 **INPUT FILES**

21635 The input files shall be text files.

21636 **ENVIRONMENT VARIABLES**

21637 The following environment variables shall affect the execution of *lp*:

21638 *LANG* Provide a default value for the internationalization variables that are unset or null.
21639 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
21640 Internationalization Variables for the precedence of internationalization variables
21641 used to determine the values of locale categories.)

21642 *LC_ALL* If set to a non-empty string value, override the values of all the other
21643 internationalization variables.

21644 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
21645 characters (for example, single-byte as opposed to multi-byte characters in
21646 arguments and input files).

21647 *LC_MESSAGES*
21648 Determine the locale that should be used to affect the format and contents of
21649 diagnostic messages written to standard error and informative messages written to
21650 standard output.

21651 *LC_TIME* Determine the format and contents of date and time strings displayed in the *lp*
21652 banner page, if any.

21653 *LPDEST* Determine the destination. If the *LPDEST* environment variable is not set, the
21654 *PRINTER* environment variable shall be used. The **–d** *dest* option takes precedence
21655 over *LPDEST*. Results are undefined when **–d** is not specified and *LPDEST*
21656 contains a value that is not a valid destination name.

21657 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

21658 *PRINTER* Determine the output device or destination. If the *LPDEST* and *PRINTER*
21659 environment variables are not set, an unspecified output device is used. The **–d**
21660 *dest* option and the *LPDEST* environment variable shall take precedence over
21661 *PRINTER*. Results are undefined when **–d** is not specified, *LPDEST* is unset, and

PRINTER contains a value that is not a valid device or destination name.

TZ Determine the timezone used to calculate date and time strings displayed in the *lp* banner page, if any. If *TZ* is unset or null, an unspecified default timezone shall be used.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The *lp* utility shall write a *request ID* to the standard output, unless **–s** is specified. The format of the message is unspecified. The request ID can be used on systems supporting the historical *cancel* and *lpstat* utilities.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All input files were processed successfully.

>0 No output device was available, or an error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *pr* and *fold* utilities can be used to achieve reasonable formatting for the implementation's default page size.

A conforming application can use one of the *file* operands only with the **–c** option or if the file is publicly readable and guaranteed to be available at the time of printing. This is because the standard gives the implementation the freedom to queue up the request for printing at some later time by a different process that might not be able to access the file.

EXAMPLES

1. To print file *file*:

   ```
   lp –c file
   ```

2. To print multiple files with headers:

   ```
   pr file1 file2 | lp
   ```

RATIONALE

The *lp* utility was designed to be a basic version of a utility that is already available in many historical implementations. The standard developers considered that it should be implementable simply as:

```
cat "$@" > /dev/lp
```

after appropriate processing of options, if that is how the implementation chose to do it and if exclusive access could be granted (so that two users did not write to the device simultaneously). Although in the future the standard developers may add other options to this utility, it should

always be able to execute with no options or operands and send the standard input to an unspecified output device.

This volume of IEEE Std 1003.1-200x makes no representations concerning the format of the printed output, except that it must be ''human-readable'' and ''non-volatile''. Thus, writing by default to a disk or tape drive or a display terminal would not qualify. (Such destinations are not prohibited when **–d** *dest*, *LPDEST*, or *PRINTER* are used, however.)

This volume of IEEE Std 1003.1-200x is worded such that a ''print job'' consisting of multiple input files, possibly in multiple copies, is guaranteed to print so that any one file is not intermixed with another, but there is no statement that all the files or copies have to print out together.

The **–c** option may imply a spooling operation, but this is not required. The utility can be implemented to wait until the printer is ready and then wait until it is finished. Because of that, there is no attempt to define a queuing mechanism (priorities, classes of output, and so on).

On some historical systems, the request ID reported on the STDOUT can be used to later cancel or find the status of a request using utilities not defined in this volume of IEEE Std 1003.1-200x.

Although the historical System V *lp* and BSD *lpr* utilities have provided similar functionality, they used different names for the environment variable specifying the destination printer. Since the name of the utility here is *lp*, *LPDEST* (used by the System V *lp* utility) was given precedence over *PRINTER* (used by the BSD *lpr* utility). Since environments of users frequently contain one or the other environment variable, the *lp* utility is required to recognize both. If this was not done, many applications would send output to unexpected output devices when users moved from system to system.

Some have commented that *lp* has far too little functionality to make it worthwhile. Requests have proposed additional options or operands or both that added functionality. The requests included:

- Wording *requiring* the output to be ''hardcopy''
- A requirement for multiple printers
- Options for supporting various page-description languages

Given that a compliant system is not required to even have a printer, placing further restrictions upon the behavior of the printer is not useful. Since hardcopy format is so application-dependent, it is difficult, if not impossible, to select a reasonable subset of functionality that should be required on all compliant systems.

The term ''unspecified'' is used in this section in lieu of ''implementation-defined'' as most known implementations would not be able to make definitive statements in their conformance documents: the existence and usage of printers is very dependent on how the system administrator configures each individual system.

Since the default destination, device type, queuing mechanisms, and acceptable forms of input are all unspecified, usage guidelines for what a conforming application can do are as follows:

- Use the command in a pipeline, or with **–c**, so that there are no permission problems and the files can be safely deleted or modified.
- Limit output to text files of reasonable line lengths and printable characters and include no device-specific formatting information, such as a page description language. The meaning of ''reasonable'' in this context can only be answered as a quality-of-implementation issue, but it should be apparent from historical usage patterns in the industry and the locale. The *pr* and *fold* utilities can be used to achieve reasonable formatting for the default page size of the

implementation.

Alternatively, the application can arrange its installation in such a way that it requires the system administrator or operator to provide the appropriate information on *lp* options and environment variable values.

At a minimum, having this utility in this volume of IEEE Std 1003.1-200x tells the industry that conforming applications require a means to print output and provides at least a command name and *LPDEST* routing mechanism that can be used for discussions between vendors, application writers, and users. The use of ‘‘should’’ in the DESCRIPTION of *lp* clearly shows the intent of the standard developers, even if they cannot mandate that all systems (such as laptops) have printers.

This volume of IEEE Std 1003.1-200x does not specify what the ownership of the process performing the writing to the output device may be. If **–c** is not used, it is unspecified whether the process performing the writing to the output device has permission to read *file* if there are any restrictions in place on who may read *file* until after it is printed. Also, if **–c** is not used, the results of deleting *file* before it is printed are unspecified.

FUTURE DIRECTIONS

None.

SEE ALSO

mailx

CHANGE HISTORY

First released in Issue 2.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the DESCRIPTION, the requirement to associate a unique request ID, and the normal generation of a banner page is added.
- In the OPTIONS section:
  - The **–d** *dest* description is expanded, but references to *lpstat* are removed.
  - The **–m**, **–o**, **–s**, **–t**, and **–w** options are added.
- In the ENVIRONMENT VARIABLES section, *LC_TIME* may now affect the execution.
- The STDOUT section is added.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

21782 **NAME**

21783 ls — list directory contents

21784 **SYNOPSIS**

21785 XSI `ls [–CFRacdilqrtu1][–H | –L ][–fgmnopsx][`*file*`...]`

21786 **DESCRIPTION**

21787 For each operand that names a file of a type other than directory or symbolic link to a directory,
21788 *ls* shall write the name of the file as well as any requested, associated information. For each
21789 operand that names a file of type directory, *ls* shall write the names of files contained within the
21790 directory as well as any requested, associated information. If one of the **–d**, **–F**, or **–l** options are
21791 specified, and one of the **–H** or **–L** options are not specified, for each operand that names a file of
21792 type symbolic link to a directory, *ls* shall write the name of the file as well as any requested,
21793 associated information. If none of the **–d**, **–F**, or **–l** options are specified, or the **–H** or **–L** options
21794 are specified, for each operand that names a file of type symbolic link to a directory, *ls* shall write
21795 the names of files contained within the directory as well as any requested, associated
21796 information.

21797 If no operands are specified, *ls* shall write the contents of the current directory. If more than one
21798 operand is specified, *ls* shall write non-directory operands first; it shall sort directory and non-
21799 directory operands separately according to the collating sequence in the current locale.

21800 The *ls* utility shall detect infinite loops; that is, entering a previously visited directory that is an
21801 ancestor of the last file encountered. When it detects an infinite loop, *ls* shall write a diagnostic
21802 message to standard error and shall either recover its position in the hierarchy or terminate.

21803 **OPTIONS**

21804 The *ls* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
21805 Utility Syntax Guidelines.

21806 The following options shall be supported:

21807 **–C** Write multi-text-column output with entries sorted down the columns, according
21808 to the collating sequence. The number of text columns and the column separator
21809 characters are unspecified, but should be adapted to the nature of the output
21810 device.

21811 **–F** Do not follow symbolic links named as operands unless the **–H** or **–L** options are
21812 specified. Write a slash (`'/'`) immediately after each pathname that is a directory,
21813 an asterisk (`'*'`) after each that is executable, a vertical bar (`'|'`) after each that is
21814 a FIFO, and an at sign (`'@'`) after each that is a symbolic link. For other file types,
21815 other symbols may be written.

21816 **–H** If a symbolic link referencing a file of type directory is specified on the command
21817 line, *ls* shall evaluate the file information and file type to be those of the file
21818 referenced by the link, and not the link itself; however, *ls* shall write the name of
21819 the link itself and not the file referenced by the link.

21820 **–L** Evaluate the file information and file type for all symbolic links (whether named
21821 on the command line or encountered in a file hierarchy) to be those of the file
21822 referenced by the link, and not the link itself; however, *ls* shall write the name of
21823 the link itself and not the file referenced by the link. When **–L** is used with **–l**, write
21824 the contents of symbolic links in the long format (see the STDOUT section).

21825 **–R** Recursively list subdirectories encountered.

21826 **–a** Write out all directory entries, including those whose names begin with a period
21827 (`'.'`). Entries beginning with a period shall not be written out unless explicitly

referenced, the –**a** option is supplied, or an implementation-defined condition shall cause them to be written.

–**c** Use time of last modification of the file status information (see **<sys/stat.h>** in the System Interfaces volume of IEEE Std 1003.1-200x) instead of last modification of the file itself for sorting (–**t**) or writing (–**l**).

–**d** Do not follow symbolic links named as operands unless the –**H** or –**L** options are specified. Do not treat directories differently than other types of files. The use of –**d** with –**R** produces unspecified results.

XSI –**f** Force each argument to be interpreted as a directory and list the name found in each slot. This option shall turn off –**l**, –**t**, –**s**, and –**r**, and shall turn on –**a**; the order is the order in which entries appear in the directory.

XSI –**g** The same as –**l**, except that the owner shall not be written.

–**i** For each file, write the file's file serial number (see *stat*( ) in the System Interfaces volume of IEEE Std 1003.1-200x).

–**l** (The letter ell.) Do not follow symbolic links named as operands unless the –**H** or –**L** options are specified. Write out in long format (see the STDOUT section). When –**l** (ell) is specified, –1 (one) shall be assumed.

XSI –**m** Stream output format; list files across the page, separated by commas.

XSI –**n** The same as –**l**, except that the owner's UID and GID numbers shall be written, rather than the associated character strings.

XSI –**o** The same as –**l**, except that the group shall not be written.

XSI –**p** Write a slash (`'/'`) after each filename if that file is a directory.

–**q** Force each instance of non-printable filename characters and <tab>s to be written as the question-mark (`'?'`) character. Implementations may provide this option by default if the output is to a terminal device.

–**r** Reverse the order of the sort to get reverse collating sequence or oldest first.

XSI –**s** Indicate the total number of file system blocks consumed by each file displayed. The block size is implementation-defined.

–**t** Sort with the primary key being time modified (most recently modified first) and the secondary key being filename in the collating sequence.

–**u** Use time of last access (see **<sys/stat.h>** in the System Interfaces volume of IEEE Std 1003.1-200x) instead of last modification of the file for sorting (–**t**) or writing (–**l**).

XSI –**x** The same as –**C**, except that the multi-text-column output is produced with entries sorted across, rather than down, the columns.

–**1** (The numeric digit one.) Force output to be one entry per line.

Specifying more than one of the options in the following mutually exclusive pairs shall not be
XSI considered an error: –**C** and –**l** (ell), –**m** and –**l** (ell), –**x** and –**l** (ell), –**C** and –**1** (one), –**H** and –**L**, –**c** and –**u**. The last option specified in each pair shall determine the output format.

21867 **OPERANDS**

21868 The following operand shall be supported:

21869 *file* A pathname of a file to be written. If the file specified is not found, a diagnostic
21870 message shall be output on standard error.

21871 **STDIN**

21872 Not used.

21873 **INPUT FILES**

21874 None.

21875 **ENVIRONMENT VARIABLES**

21876 The following environment variables shall affect the execution of *ls*:

21877 *COLUMNS* Determine the user's preferred column position width for writing multiple text-
21878 column output. If this variable contains a string representing a decimal integer, the
21879 *ls* utility shall calculate how many pathname text columns to write (see **–C**) based
21880 on the width provided. If *COLUMNS* is not set or invalid, an implementation-
21881 defined number of column positions shall be assumed, based on the
21882 implementation's knowledge of the output device. The column width chosen to
21883 write the names of files in any given directory shall be constant. Filenames shall
21884 not be truncated to fit into the multiple text-column output.

21885 *LANG* Provide a default value for the internationalization variables that are unset or null.
21886 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
21887 Internationalization Variables for the precedence of internationalization variables
21888 used to determine the values of locale categories.)

21889 *LC_ALL* If set to a non-empty string value, override the values of all the other
21890 internationalization variables.

21891 *LC_COLLATE*
21892 Determine the locale for character collation information in determining the
21893 pathname collation sequence.

21894 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
21895 characters (for example, single-byte as opposed to multi-byte characters in
21896 arguments) and which characters are defined as printable (character class **print**).

21897 *LC_MESSAGES*
21898 Determine the locale that should be used to affect the format and contents of
21899 diagnostic messages written to standard error.

21900 *LC_TIME* Determine the format and contents for date and time strings written by *ls*.

21901 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

21902 *TZ* Determine the timezone for date and time strings written by *ls*. If *TZ* is unset or
21903 null, an unspecified default timezone shall be used.

21904 **ASYNCHRONOUS EVENTS**

21905 Default.

21906 **STDOUT**

21907 The default format shall be to list one entry per line to standard output; the exceptions are to
21908 XSI terminals or when one of the **–C**, **–m**, or **–x** options is specified. If the output is to a terminal, the
21909 format is implementation-defined.

21910 XSI When –**m** is specified, the format used shall be:

21911 `"%s, %s, ...\n", <filename1>, <filename2>`

21912 where the largest number of filenames shall be written without exceeding the length of the line.

21913 If the –**i** option is specified, the file's file serial number (see **<sys/stat.h>** in the System Interfaces
21914 volume of IEEE Std 1003.1-200x) shall be written in the following format before any other output
21915 for the corresponding entry:

21916 `%u ", <file serial number>`

21917 If the –**l** option is specified without –**L**, the following information shall be written:

```
21918 "%s %u %s %s %u %s %s\n", <file mode>, <number of links>,
21919     <owner name>, <group name>, <number of bytes in the file>,
21920     <date and time>, <pathname>
```

21921 If the file is a symbolic link, this information shall be about the link itself and the <*pathname*>
21922 field shall be of the form:

21923 `"%s –> %s", <pathname of link>, <contents of link>`

21924 If both –**l** and –**L** are specified, the following information shall be written:

```
21925 "%s %u %s %s %u %s %s\n", <file mode>, <number of links>,
21926     <owner name>, <group name>, <number of bytes in the file>,
21927     <date and time>, <pathname of link>
```

21928 where all fields except <*pathname of link*> shall be for the file resolved from the symbolic link.

21929 XSI The –**g**, –**n**, and –**o** options use the same format as –**l**, but with omitted items and their
21930 associated <blank>s. See the OPTIONS section.

21931 XSI In both the preceding –**l** forms, if <*owner name*> or <*group name*> cannot be determined, or if –**n** |
21932 is given, they shall be replaced with their associated numeric values using the format %u.

21933 The <*date and time*>, field shall contain the appropriate date and timestamp of when the file was
21934 last modified. In the POSIX locale, the field shall be the equivalent of the output of the following
21935 *date* command:

21936 `date "+%b %e %H:%M"`

21937 if the file has been modified in the last six months, or:

21938 `date "+%b %e %Y"`

21939 (where two <space>s are used between %e and %Y) if the file has not been modified in the last six
21940 months or if the modification date is in the future, except that, in both cases, the final <newline>
21941 produced by *date* shall not be included and the output shall be as if the *date* command were
21942 executed at the time of the last modification date of the file rather than the current time. When
21943 the *LC_TIME* locale category is not set to the POSIX locale, a different format and order of
21944 presentation of this field may be used.

21945 If the file is a character special or block special file, the size of the file may be replaced with
21946 implementation-defined information associated with the device in question.

21947 If the pathname was specified as a *file* operand, it shall be written as specified.

21948 XSI The file mode written under the –**l**, –**g**, –**n**, and –**o** options shall consist of the following format:

```
21949 "%c%s%s%s%c", <entry type>, <owner permissions>,
21950     <group permissions>, <other permissions>,
```

21951 *<optional alternate access method flag>*

21952 The *<optional alternate access method flag>* shall be a single <space> if there is no alternate or
21953 additional access control method associated with the file; otherwise, a printable character shall
21954 be used.

21955 The *<entry type>* character shall describe the type of file, as follows:

21956 d Directory.

21957 b Block special file.

21958 c Character special file.

21959 l (ell) Symbolic link.

21960 p FIFO.

21961 – Regular file.

21962 Implementations may add other characters to this list to represent other implementation-defined
21963 file types.

21964 The next three fields shall be three characters each:

21965 *<owner permissions>*
21966 Permissions for the file owner class (see the Base Definitions volume of
21967 IEEE Std 1003.1-200x, Section 4.4, File Access Permissions).

21968 *<group permissions>*
21969 Permissions for the file group class.

21970 *<other permissions>*
21971 Permissions for the file other class.

21972 Each field shall have three character positions:

21973 1. If 'r', the file is readable; if '–', the file is not readable.

21974 2. If 'w', the file is writable; if '–', the file is not writable.

21975 3. The first of the following that applies:

21976 S If in *<owner permissions>*, the file is not executable and set-user-ID mode is set. If in
21977 *<group permissions>*, the file is not executable and set-group-ID mode is set.

21978 s If in *<owner permissions>*, the file is executable and set-user-ID mode is set. If in
21979 *<group permissions>*, the file is executable and set-group-ID mode is set.

21980 x The file is executable or the directory is searchable.

21981 – None of the attributes of 'S', 's', or 'x' applies.

21982 Implementations may add other characters to this list for the third character position. Such
21983 additions shall, however, be written in lowercase if the file is executable or searchable, and
21984 in uppercase if it is not.

21985 XSI If any of the **–l**, **–g**, **–n**, **–o**, or **–s** options is specified, each list of files within the directory shall be
21986 preceded by a status line indicating the number of file system blocks occupied by files in the
21987 directory in 512-byte units, rounded up to the next integral number of units, if necessary. In the
21988 POSIX locale, the format shall be:

21989 "total %u\n", *<number of units in the directory>*

If more than one directory, or a combination of non-directory files and directories are written, either as a result of specifying multiple operands, or the **–R** option, each list of files within a directory shall be preceded by:

`"\n%s:\n", <directory name>`

If this string is the first thing to be written, the first <newline> shall not be written. This output shall precede the number of units in the directory.

XSI If the **–s** option is given, each file shall be written with the number of blocks used by the file. Along with **–C**, **–1**, **–m**, or **–x**, the number and a <space> shall precede the filename; with **–g**, **–l**, **–n**, or **–o**, they shall precede each line describing a file.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Many implementations use the equal sign (`'='`) to denote sockets bound to the file system for the **–F** option. Similarly, many historical implementations use the `'s'` character to denote sockets as the entry type characters for the **–l** option.

It is difficult for an application to use every part of the file modes field of *ls* **–l** in a portable manner. Certain file types and executable bits are not guaranteed to be exactly as shown, as implementations may have extensions. Applications can use this field to pass directly to a user printout or prompt, but actions based on its contents should generally be deferred, instead, to the *test* utility.

The output of *ls* (with the **–l** and related options) contains information that logically could be used by utilities such as *chmod* and *touch* to restore files to a known state. However, this information is presented in a format that cannot be used directly by those utilities or be easily translated into a format that can be used. A character has been added to the end of the permissions string so that applications at least have an indication that they may be working in an area they do not understand instead of assuming that they can translate the permissions string into something that can be used. Future issues or related documents may define one or more specific characters to be used based on different standard additional or alternative access control mechanisms.

As with many of the utilities that deal with filenames, the output of *ls* for multiple files or in one of the long listing formats must be used carefully on systems where filenames can contain embedded white space. Systems and system administrators should institute policies and user training to limit the use of such filenames.

The number of disk blocks occupied by the file that it reports varies depending on underlying file system type, block size units reported, and the method of calculating the number of blocks.

On some file system types, the number is the actual number of blocks occupied by the file (counting indirect blocks and ignoring holes in the file); on others it is calculated based on the file size (usually making an allowance for indirect blocks, but ignoring holes).

EXAMPLES

An example of a small directory tree being fully listed with *ls* –**laRF a** in the POSIX locale:

```
total 11
drwxr-xr-x   3 hlj      prog          64 Jul  4 12:07 ./
drwxrwxrwx   4 hlj      prog        3264 Jul  4 12:09 ../
drwxr-xr-x   2 hlj      prog          48 Jul  4 12:07 b/
-rwxr--r--   1 hlj      prog         572 Jul  4 12:07 foo*

a/b:
total 4
drwxr-xr-x   2 hlj      prog          48 Jul  4 12:07 ./
drwxr-xr-x   3 hlj      prog          64 Jul  4 12:07 ../
-rw-r--r--   1 hlj      prog         700 Jul  4 12:07 bar
```

RATIONALE

Some historical implementations of the *ls* utility show all entries in a directory except dot and dot-dot when a superuser invokes *ls* without specifying the –**a** option. When ''normal'' users invoke *ls* without specifying –**a**, they should not see information about any files with names beginning with period unless they were named as *file* operands.

Implementations are expected to traverse arbitrary depths when processing the –**R** option. The only limitation on depth should be based on running out of physical storage for keeping track of untraversed directories.

The –**1** (one) option is currently found in BSD and BSD-derived implementations only. It is required in this volume of IEEE Std 1003.1-200x so that conforming applications might ensure that output is one entry per line, even if the output is to a terminal.

Generally, this volume of IEEE Std 1003.1-200x is silent about what happens when options are given multiple times. In the cases of –**C**, –**l**, and –**1**, however, it does specify the results of these overlapping options. Since *ls* is one of the most aliased commands, it is important that the implementation perform intuitively. For example, if the alias were:

```
alias ls="ls –C"
```

and the user typed *ls* –**1**, single-text-column output should result, not an error.

The BSD *ls* provides a –**A** option (like –**a**, but dot and dot-dot are not written out). The small difference from –**a** did not seem important enough to require both.

Implementations may make –**q** the default for terminals to prevent trojan horse attacks on terminals with special escape sequences. This is not required because:

- Some control characters may be useful on some terminals; for example, a system might write them as `"\001"` or `"^A"`.
- Special behavior for terminals is not relevant to application portability.

An early proposal specified that the optional alternate access method flag had to be `'+'` if there was an alternate access method used on the file or <space> if there was not. This was changed to be <space> if there is not and a single printable character if there is. This was done for three reasons:

1. There are historical implementations using characters other than `'+'`.

2. There are implementations that vary this character used in that position to distinguish between various alternate access methods in use.

3. The standard developers did not want to preclude futures specifications that might need a way to specify more than one alternate access method.

Nonetheless, implementations providing a single alternate access method are encouraged to use '+'.

In an early proposal, the units used to specify the number of blocks occupied by files in a directory in an *ls* **–l** listing was implementation-defined. This was because BSD systems have historically used 1 024-byte units and System V systems have historically used 512-byte units. It was pointed out by BSD developers that their system has used 512-byte units in some places and 1 024-byte units in other places. (System V has consistently used 512.) Therefore, this volume of IEEE Std 1003.1-200x usually specifies 512. Future releases of BSD are expected to consistently provide 512 bytes as a default with a way of specifying 1 024-byte units where appropriate.

The <*date and time*> field in the **–l** format is specified only for the POSIX locale. As noted, the format can be different in other locales. No mechanism for defining this is present in this volume of IEEE Std 1003.1-200x, as the appropriate vehicle is a messaging system; that is, the format should be specified as a ''message''.

FUTURE DIRECTIONS

The **–s** uses implementation-defined units and cannot be used portably; it may be withdrawn in a future issue.

SEE ALSO

chmod, *find*, the System Interfaces volume of IEEE Std 1003.1-200x, **<sys/stat.h>**

CHANGE HISTORY

First released in Issue 2.

Issue 5

Second FUTURE DIRECTION added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- In the **–F** option, other symbols are allowed for other file types.

Treatment of symbolic links is added, as defined in the IEEE P1003.2b draft standard.

22110 **NAME**

22111 m4 — macro processor (**DEVELOPMENT**)

22112 **SYNOPSIS**

22113 XSI `m4 [-s][-D name[=val]]...[-U name]... file...`

22114

22115 **DESCRIPTION**

22116 The *m4* utility is a macro processor that shall read one or more text files, process them according
22117 to their included macro statements, and write the results to standard output.

22118 **OPTIONS**

22119 The *m4* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
22120 12.2, Utility Syntax Guidelines, except that the order of the **-D** and **-U** options shall be
22121 significant.

22122 The following options shall be supported:

22123 **-s** Enable line synchronization output for the *c99* preprocessor phase (that is, **#line**
22124 directives).

22125 **-D** *name*[=*val*]
22126 Define *name* to *val* or to null if =*val* is omitted.

22127 **-U** *name* Undefine *name*.

22128 **OPERANDS**

22129 The following operand shall be supported:

22130 *file* A pathname of a text file to be processed. If no *file* is given, or if it is '-', the
22131 standard input shall be read.

22132 **STDIN**

22133 The standard input shall be a text file that is used if no *file* operand is given, or if it is '-'.

22134 **INPUT FILES**

22135 The input file named by the *file* operand shall be a text file.

22136 **ENVIRONMENT VARIABLES**

22137 The following environment variables shall affect the execution of *m4*:

22138 *LANG* Provide a default value for the internationalization variables that are unset or null.
22139 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
22140 Internationalization Variables for the precedence of internationalization variables
22141 used to determine the values of locale categories.)

22142 *LC_ALL* If set to a non-empty string value, override the values of all the other
22143 internationalization variables.

22144 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
22145 characters (for example, single-byte as opposed to multi-byte characters in
22146 arguments and input files).

22147 *LC_MESSAGES*
22148 Determine the locale that should be used to affect the format and contents of
22149 diagnostic messages written to standard error.

22150 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall be the same as the input files, after being processed for macro expansion.

STDERR

The standard error shall be used to display strings with the **errprint** macro, macro tracing enabled by the **traceon** macro, the defined text for macros written by the **dumpdef** macro, or for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

The *m4* utility shall compare each token from the input against the set of built-in and user-defined macros. If the token matches the name of a macro, then the token shall be replaced by the macro's defining text, if any, and rescanned for matching macro names. Once no portion of the token matches the name of a macro, it shall be written to standard output. Macros may have arguments, in which case the arguments shall be substituted into the defining text before it is rescanned.

Macro calls have the form:

name(*arg1, arg2, ..., argn*)

Macro names shall consist of letters, digits, and underscores, where the first character is not a digit. Tokens not of this form shall not be treated as macros.

The application shall ensure that the left parenthesis immediately follows the name of the macro. If a token matching the name of a macro is not followed by a left parenthesis, it is handled as a use of that macro without arguments.

If a macro name is followed by a left parenthesis, its arguments are the comma-separated tokens between the left parenthesis and the matching right parenthesis. Unquoted <blank>s and <newline>s preceding each argument shall be ignored. All other characters, including trailing <blank>s and <newline>s, are retained. Commas enclosed between left and right parenthesis characters do not delimit arguments.

Arguments are positionally defined and referenced. The string "$1" in the defining text shall be replaced by the first argument. Systems shall support at least nine arguments; only the first nine can be referenced, using the strings "$1" to "$9", inclusive. The string "$0" is replaced with the name of the macro. The string "$#" is replaced by the number of arguments as a string. The string "$*" is replaced by a list of all of the arguments, separated by commas. The string "$@" is replaced by a list of all of the arguments separated by commas, and each argument is quoted using the current left and right quoting strings.

If fewer arguments are supplied than are in the macro definition, the omitted arguments are taken to be null. It is not an error if more arguments are supplied than are in the macro definition.

No special meaning is given to any characters enclosed between matching left and right quoting strings, but the quoting strings are themselves discarded. By default, the left quoting string consists of a grave accent (`) and the right quoting string consists of an acute accent (`); see also the **changequote** macro.

Comments are written but not scanned for matching macro names; by default, the begin-comment string consists of the number sign character and the end-comment string consists of a

<newline>. See also the **changecom** and **dnl** macros.

The *m4* utility shall make available the following built-in macros. They can be redefined, but once this is done the original meaning is lost. Their values shall be null unless otherwise stated. In the descriptions below, the term *defining text* refers to the value of the macro: the second argument to the **define** macro, among other things. Except for the first argument to the **eval** macro, all numeric arguments to built-in macros shall be interpreted as decimal values. The string values produced as the defining text of the **decr**, **divnum**, **incr**, **index**, **len**, and **sysval** built-in macros shall be in the form of a decimal-constant as defined in the C language.

changecom The **changecom** macro shall set the begin-comment and end-comment strings. With no arguments, the comment mechanism shall be disabled. With a single argument, that argument shall become the begin-comment string and the <newline> shall become the end-comment string. With two arguments, the first argument shall become the begin-comment string and the second argument shall become the end-comment string. Systems shall support comment strings of at least five characters.

changequote The **changequote** macro shall set the begin-quote and end-quote strings. With no arguments, the quote strings shall be set to the default values (that is, ‘ ’). With a single argument, that argument shall become the begin-quote string and the <newline> shall become the end-quote string. With two arguments, the first argument shall become the begin-quote string and the second argument shall become the end-quote string. Systems shall support quote strings of at least five characters.

decr The defining text of the **decr** macro shall be its first argument decremented by 1. It shall be an error to specify an argument containing any non-numeric characters.

define The second argument shall become the defining text of the macro whose name is the first argument.

defn The defining text of the **defn** macro shall be the quoted definition (using the current quoting strings) of its arguments.

divert The *m4* utility maintains nine temporary buffers, numbered 1 to 9, inclusive. When the last of the input has been processed, any output that has been placed in these buffers shall be written to standard output in buffer-numerical order. The **divert** macro shall divert future output to the buffer specified by its argument. Specifying no argument or an argument of 0 shall resume the normal output process. Output diverted to a stream other than 0 to 9 shall be discarded. It shall be an error to specify an argument containing any non-numeric characters.

divnum The defining text of the **divnum** macro shall be the number of the current output stream as a string.

dnl The **dnl** macro shall cause *m4* to discard all input characters up to and including the next <newline>.

dumpdef The **dumpdef** macro shall write the defined text to standard error for each of the macros specified as arguments, or, if no arguments are specified, for all macros.

errprint The **errprint** macro shall write its arguments to standard error.

eval The **eval** macro shall evaluate its first argument as an arithmetic expression, using 32-bit signed integer arithmetic. All of the C-language operators shall be supported, except for:

22242 `[]`
22243 `->`
22244 `++`
22245 `--`
22246 `(`*`type`*`)`
22247 `unary *`
22248 *`sizeof`*
22249 `,`
22250 `.`
22251 `?:`
22252 `unary &`

22253 and all assignment operators. It shall be an error to specify any of these operators. |
22254 Precedence and associativity shall be as in the ISO C standard. Systems shall |
22255 support octal and hexadecimal numbers as in the ISO C standard. The second |
22256 argument, if specified, shall set the radix for the result; the default is 10. The third |
22257 argument, if specified, sets the minimum number of digits in the result. It shall be |
22258 an error to specify the second or third argument containing any non-numeric |
22259 characters. |

22260 **ifdef** If the first argument to the **ifdef** macro is defined, the defining text shall be the |
22261 second argument. Otherwise, the defining text shall be the third argument, if |
22262 specified, or the null string, if not. |

22263 **ifelse** The **ifelse** macro takes three or more arguments. If the first two arguments |
22264 compare as equal strings (after macro expansion of both arguments), the defining |
22265 text shall be the third argument. If the first two arguments do not compare as |
22266 equal strings and there are three arguments, the defining text shall be null. If the |
22267 first two arguments do not compare as equal strings and there are four or five |
22268 arguments, the defining text shall be the fourth argument. If the first two |
22269 arguments do not compare as equal strings and there are six or more arguments, |
22270 the first three arguments shall be discarded and processing shall restart with the |
22271 remaining arguments. |

22272 **include** The defining text for the **include** macro shall be the contents of the file named by |
22273 the first argument. It shall be an error if the file cannot be read. |

22274 **incr** The defining text of the **incr** macro shall be its first argument incremented by 1. It |
22275 shall be an error to specify an argument containing any non-numeric characters. |

22276 **index** The defining text of the **index** macro shall be the first character position (as a |
22277 string) in the first argument where a string matching the second argument begins |
22278 (zero origin), or –1 if the second argument does not occur.

22279 **len** The defining text of the **len** macro shall be the length (as a string) of the first |
22280 argument. |

22281 **m4exit** Exit from the *m4* utility. If the first argument is specified, it is the exit code. The |
22282 default is zero. It shall be an error to specify an argument containing any non- |
22283 numeric characters. |

22284 **m4wrap** The first argument shall be processed when EOF is reached. If the **m4wrap** macro |
22285 is used multiple times, the arguments specified shall be processed in the order in |
22286 which the **m4wrap** macros were processed. |

22287 **maketemp** The defining text shall be the first argument, with any trailing `'X'` characters |
22288 replaced with the current process ID as a string.

popdef The **popdef** macro shall delete the current definition of its arguments, replacing that definition with the previous one. If there is no previous definition, the macro is undefined.

pushdef The **pushdef** macro shall be equivalent to the **define** macro with the exception that it shall preserve any current definition for future retrieval using the **popdef** macro.

shift The defining text for the **shift** macro shall be all of its arguments except for the first one.

sinclude The **sinclude** macro shall be equivalent to the **include** macro, except that it shall not be an error if the file is inaccessible.

substr The defining text for the **substr** macro shall be the substring of the first argument beginning at the zero-offset character position specified by the second argument. The third argument, if specified, shall be the number of characters to select; if not specified, the characters from the starting point to the end of the first argument shall become the defining text. It shall not be an error to specify a starting point beyond the end of the first argument and the defining text shall be null. It shall be an error to specify an argument containing any non-numeric characters.

syscmd The **syscmd** macro shall interpret its first argument as a shell command line. The defining text shall be the string result of that command. No output redirection shall be performed by the *m4* utility. The exit status value from the command can be retrieved using the **sysval** macro.

sysval The defining text of the **sysval** macro shall be the exit value of the utility last invoked by the **syscmd** macro (as a string).

traceon The **traceon** macro shall enable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros. The trace output shall be written to standard error in an unspecified format.

traceoff The **traceoff** macro shall disable tracing for the macros specified as arguments, or, if no arguments are specified, for all macros.

translit The defining text of the **translit** macro shall be the first argument with every character that occurs in the second argument replaced with the corresponding character from the third argument.

undefine The **undefine** macro shall delete all definitions (including those preserved using the **pushdef** macro) of the macros named by its arguments.

undivert The **undivert** macro shall cause immediate output of any text in temporary buffers named as arguments, or all temporary buffers if no arguments are specified. Buffers can be undiverted into other temporary buffers. Undiverting shall discard the contents of the temporary buffer. It shall be an error to specify an argument containing any non-numeric characters.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred

If the **m4exit** macro is used, the exit value can be specified by the input file.

22331 **CONSEQUENCES OF ERRORS**

22332 Default.

22333 **APPLICATION USAGE**

22334 The **defn** macro is useful for renaming macros, especially built-ins.

22335 **EXAMPLES**

22336 An example of a single *m4* input file capable of generating two output files follows. The file
22337 **file1.m4** could contain lines such as:

```
22338 if(VER, 1, do_something)
22339 if(VER, 2, do_something)
```

22340 The makefile for the program might include:

```
22341 file1.1.c : file1.m4
22342             m4 –D VER=1 file1.m4 > file1.1.c
22343             ...
22344 file1.2.c : file1.m4
22345             m4 –D VER=2 file1.m4 > file1.2.c
22346             ...
```

22347 The **–U** option can be used to undefine **VER**. If **file1.m4** contains:

```
22348 if(VER, 1,  do_something)
22349 if(VER, 2,  do_something)
22350 ifndef(VER, do_something)
```

22351 then the makefile would contain:

```
22352 file1.0.c : file1.m4
22353             m4 –U VER file1.m4 > file1.0.c
22354             ...
22355 file1.1.c : file1.m4
22356             m4 –D VER=1 file1.m4 > file1.1.c
22357             ...
22358 file1.2.c : file1.m4
22359             m4 –D VER=2 file1.m4 > file1.2.c
22360             ...
```

22361 **RATIONALE**

22362 None.

22363 **FUTURE DIRECTIONS**

22364 None.

22365 **SEE ALSO**

22366 *c99*

22367 **CHANGE HISTORY**

22368 First released in Issue 2.

22369 **Issue 5**

22370 The phrase ‘‘the defined text for macros written by the **dumpdef** macro’’ is added to the
22371 description of STDERR, and the description of **dumpdef** is updated to indicate that output is
22372 written to standard error. The description of **eval** is updated to indicate that the list of excluded
22373 C operators excludes unary ’&’ and ’.’. In the description of **ifdef**, the phrase ‘‘and it is not
22374 defined to be zero’’ is deleted.

Issue 6

In the EXTENDED DESCRIPTION, the **eval** text is updated to include a '&' character in the excepted list.

The EXTENDED DESCRIPTION of **divert** is updated to clarify that there are only nine diversion buffers.

The normative text is reworded to avoid use of the term ''must'' for application requirements.

The Open Group Base Resolution bwg2000-006 is applied.

NAME

mailx — process messages

SYNOPSIS

Send Mode

```
mailx [–s subject] address...
```

Receive Mode

```
mailx –e
mailx [–HiNn][–F][–u user]
mailx –f[–HiNn][–F][file]
```

DESCRIPTION

The *mailx* utility provides a message sending and receiving facility. It has two major modes, selected by the options used: Send Mode and Receive Mode.

On systems that do not support the User Portability Utilities option, an application using *mailx* shall have the ability to send messages in an unspecified manner (Send Mode). Unless the first character of one or more lines is tilde (`'~'`), all characters in the input message shall appear in the delivered message, but additional characters may be inserted in the message before it is retrieved.

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

Send Mode

Send Mode can be used by applications or users to send messages from the text in standard input.

Receive Mode

Receive Mode is more oriented to interactive users. Mail can be read and sent in this interactive mode.

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, collectively called the system *mailbox* for that user. When *mailx* is invoked in Receive Mode, the system mailbox shall be the default place to find new mail. 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 is normally located in the directory referred to by the *HOME* environment variable (see *MBOX* in the ENVIRONMENT VARIABLES section 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 section as simply ‘‘mailboxes’’, unless more specific identification is required.

22420 **OPTIONS**

22421 The *mailx* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
22422 12.2, Utility Syntax Guidelines.

22423 The following options shall be supported. (Only the **–s** *subject* option shall be required on all
22424 systems. The other options are required only on systems supporting the User Portability Utilities
22425 option.)

22426 **–e** Test for the presence of mail in the system mailbox. The *mailx* utility shall write
22427 nothing and exit with a successful return code if there is mail to read.

22428 **–f** Read messages from the file named by the *file* operand instead of the system
22429 mailbox. (See also **folder**.) If no *file* operand is specified, read messages from the
22430 **mbox** instead of the system mailbox.

22431 **–F** Record the message in a file named after the first recipient. The name is the login-
22432 name portion of the address found first on the **To:** line in the mail header.
22433 Overrides the **record** variable, if set (see **Internal Variables in mailx** (on page
22434 2792).)

22435 **–H** Write a header summary only.

22436 **–i** Ignore interrupts. (See also **ignore**).

22437 **–n** Do not initialize from the system default start-up file. See the EXTENDED
22438 DESCRIPTION section.

22439 **–N** Do not write an initial header summary.

22440 **–s** *subject* Set the **Subject** header field to *subject*. All characters in the *subject* string shall
22441 appear in the delivered message. The results are unspecified if *subject* is longer
22442 than {LINE_MAX} – 10 bytes or contains a <newline>.

22443 **–u** *user* Read the system mailbox of the login name *user*. This shall only be successful if
22444 the invoking user has the appropriate privileges to read the system mailbox of that
22445 user.

22446 **OPERANDS**

22447 The following operands shall be supported:

22448 *address* Addressee of message. When **–n** is specified and no user start-up files are accessed
22449 (see the EXTENDED DESCRIPTION section), the user or application shall ensure
22450 this is an address to pass to the mail delivery system. Any system or user start-up
22451 files may enable aliases (see **alias** under **Commands in mailx** (on page 2795)) that
22452 may modify the form of *address* before it is passed to the mail delivery system.

22453 *file* A pathname of a file to be read instead of the system mailbox when **–f** is specified.
22454 The meaning of the *file* option-argument shall be affected by the contents of the
22455 **folder** internal variable; see **Internal Variables in mailx** (on page 2792).

22456 **STDIN**

22457 When *mailx* is invoked in Send Mode (the first synopsis line), standard input shall be the
22458 message to be delivered to the specified addresses. When in Receive Mode, user commands shall |
22459 be accepted from *stdin*. If the User Portability Utilities option is not supported, standard input |
22460 lines beginning with a tilde (′~′) character produce unspecified results.

22461 If the User Portability Utilities option is supported, then in both Send and Receive Modes,
22462 standard input lines beginning with the escape character (usually tilde (′~′)) shall affect |
22463 processing as described in **Command Escapes in mailx** (on page 2803). |

INPUT FILES

When *mailx* is used as described by this volume of IEEE Std 1003.1-200x, the *file* option-argument (see the **−f** option) and the **mbox** shall be text files containing mail messages, formatted as described in the OUTPUT FILES section. The nature of the system mailbox is unspecified; it need not be a file.

ENVIRONMENT VARIABLES

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

DEAD Determine the 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**.

EDITOR Determine the name of a utility to invoke when the **edit** (see **Commands in mailx** (on page 2795)) or **~e** (see **Command Escapes in mailx** (on page 2803)) command is used. The default editor is unspecified. XSI On XSI-conformant systems it is *ed*. The effects of this variable are unspecified if the User Portability Utilities option is not supported.

HOME Determine the pathname of the user's home directory.

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the handling of case-insensitive address and header-field comparisons.

LC_TIME Determine the format and contents of the date and time strings written by *mailx*.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

LISTER Determine a string representing the command for writing the contents of the **folder** directory to standard output when the **folders** command is given (see **folders** in **Commands in mailx** (on page 2795)). Any string acceptable as a *command_string* operand to the *sh* **−c** command shall be valid. If this variable is null or not set, the output command shall be *ls*. The effects of this variable are unspecified if the User Portability Utilities option is not supported.

MAILRC Determine the pathname of the start-up file. The default shall be **.mailrc** in the directory referred to by the *HOME* environment variable. 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**.

MBOX Determine 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. The effects of this variable are unspecified if the User Portability Utilities option is not supported.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

PAGER Determine a string representing an output filtering or pagination command for writing the output to the terminal. Any string acceptable as a *command_string* operand to the *sh* **–c** command 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 **Internal Variables in mailx** (on page 2792). If the *PAGER* variable is null or not set, the paginator shall be either *more* or another paginator utility documented in the system documentation. The effects of this variable are unspecified if the User Portability Utilities option is not supported.

SHELL Determine the name of a preferred command interpreter. The default shall be *sh*. The effects of this variable are unspecified if the User Portability Utilities option is not supported.

TERM Determine the name of the terminal type, to indicate in an unspecified manner, if the internal variable **screen** is not specified, the number of lines in a screenful of headers. If *TERM* is not set or is set to null, an unspecified default terminal type shall be used and the value of a screenful is unspecified. The effects of this variable are unspecified if the User Portability Utilities option is not supported.

TZ This variable may determine the timezone used to calculate date and time strings written by *mailx*. If *TZ* is unset or null, an unspecified default timezone shall be used.

VISUAL Determine a pathname of a utility to invoke when the **visual** command (see **Commands in mailx** (on page 2795)) or **~v** command-escape (see **Command Escapes in mailx** (on page 2803)) is used. If this variable is null or not set, the full-screen editor shall be *vi*. The effects of this variable are unspecified if the User Portability Utilities option is not supported.

ASYNCHRONOUS EVENTS

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

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, *mailx* shall write "@\n", discard the current input line, and continue processing, bypassing the message-abort mechanism described in item 2b.
   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 typed, shall be required to abort the mail message. If in Receive 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 non-zero status.

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

i. 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.

ii. 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.

STDOUT

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

STDERR

The standard error shall be used only for diagnostic messages.

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 as described in this volume of IEEE Std 1003.1-200x, these files shall be text files, formatted as follows:

```
line beginning with From<space>
[one or more header-lines; see Commands in mailx (on page 2795)]
empty line
[zero or more body lines
empty line]
[line beginning with From<space>...]
```

where each message begins with the **From <space>** line shown, preceded by the beginning of the file or an empty line. (The **From <space>** line is considered to be part of the message header, but not one of the header-lines referred to in **Commands in mailx** (on page 2795); thus, it shall not be affected by the **discard**, **ignore**, or **retain** commands.) The formats of the remainder of the **From <space>** line and any additional header lines are unspecified, except that none shall be empty. The format of a message body line is also unspecified, except that no line following an empty line shall start with **From <space>**; *mailx* shall modify any such user-entered message body lines (following an empty line and beginning with **From <space>**) by adding one or more characters to precede the `'F'`; it may add these characters to **From <space>** lines that are not preceded by an empty line.

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*.

EXTENDED DESCRIPTION

The entire EXTENDED DESCRIPTION section shall apply only to implementations supporting the User Portability Utilities option.

The *mailx* utility cannot guarantee support for all character encodings in all circumstances. For example, inter-system mail may be restricted to 7-bit data by the underlying network, 8-bit data need not be portable to non-internationalized systems, and so on. Under these circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used.

When *mailx* is invoked using one of the Receive Mode synopsis forms, it shall write a page of header-summary lines (if **–N** was not specified and there are messages, see below), followed by a prompt indicating that *mailx* can accept regular commands (see **Commands in mailx** (on page 2795)); this is termed *command mode*. The page of header-summary lines shall contain the first new message if there are new messages, or the first unread message if there are unread messages, or the first message. 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* shall be in input mode. This input mode shall also be entered when using one of the Receive Mode synopsis forms and a reply or new message is composed using the **reply**, **Reply**, **followup**, **Followup**, or **mail** commands and standard input is a terminal. 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 **Commands in mailx** (on page 2795) for a summary of these commands. It is unspecified what effect these commands will have if standard input is not a terminal when a message is entered using either the Send Mode synopsis, or the Read Mode commands **reply**, **Reply**, **followup**, **Followup**, or **mail**.

Note: For notational convenience, this section 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 *mailx* **set** and **unset** commands.

Regular commands are of the form:

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

If no *command* is specified in command mode, **next** 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 shall affect 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, which shall be marked by a `'>'` at the beginning of a line in the header summary. When *mailx* is invoked using one of the Receive Mode synopsis forms, the current message shall be the first new message, if there is a new message, or the first unread message if there is an unread message, or the first message if there are any messages, or unspecified if there are no messages in the mailbox. Each command that takes an optional list of messages (*msglist*) or an optional single message (*message*) on which to operate shall leave the current message set to the highest-numbered message of the messages specified, unless the command deletes messages, in which case the current message shall be set to the first undeleted message (that is, a message not in the deleted state) after the highest-numbered message deleted by the command, if one exists, or the first undeleted message before the highest-numbered message deleted by the command, if one exists, or to an unspecified value if there are no remaining undeleted messages. All messages shall be 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. Messages that are in the **mbox** or in a secondary mailbox and in state

22650 *read* when *mailx* quits shall be retained in their current location.

22651 *deleted* The message has been processed by one of the following commands: **delete**, **dp**,
22652 **dt**. Messages in state *deleted* when *mailx* quits shall be deleted. Deleted messages
22653 shall be ignored until *mailx* quits or changes mailboxes or they are specified to the
22654 undelete command; for example, the message specification /*string* shall only
22655 search the subject lines of messages that have not yet been deleted, unless the
22656 command operating on the list of messages is **undelete**. No deleted message or
22657 deleted message header shall be displayed by any *mailx* command other than
22658 **undelete**.

22659 *preserved* The message has been processed by a **preserve** command. When *mailx* quits, the
22660 message shall be retained in its current location.

22661 *saved* The message has been processed by one of the following commands: **save** or
22662 **write**. If the current mailbox is the system mailbox, and the internal variable
22663 **keepsave** is set, messages in the state saved shall be saved to the file designated by
22664 the *MBOX* variable (see the ENVIRONMENT VARIABLES section). If the current
22665 mailbox is the system mailbox, messages in the state *saved* shall be deleted from
22666 the current mailbox, when the **quit** or **file** command is used to exit the current
22667 mailbox.

22668 The header-summary line for each message shall indicate the state of the message.

22669 Many commands take an optional list of messages (*msglist*) on which to operate, which defaults
22670 to the current message. A *msglist* is a list of message specifications separated by <blank>s, which
22671 can include:

22672 `n` Message number *n*.

22673 `+` The next undeleted message, or the next deleted message for the **undelete** command.

22674 – The next previous undeleted message, or the next previous deleted message for the
22675 **undelete** command.

22676 `.` The current message.

22677 `^` The first undeleted message, or the first deleted message for the **undelete** command.

22678 `$` The last message.

22679 `*` All messages.

22680 `n-m` An inclusive range of message numbers.

22681 *address* All messages from *address*; any address as shown in a header summary shall be
22682 matchable in this form.

22683 /*string* All messages with *string* in the subject line (case ignored).

22684 `:c` All messages of type *c*, where *c* shall be one of:

22685 `d` Deleted messages.

22686 `n` New messages.

22687 `o` Old messages (any not in state *read* or *new*).

22688 `r` Read messages.

22689 `u` Unread messages.

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

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

Start-Up in mailx

At start-up 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 start-up file, unless the **–n** option is given, to initialize any internal *mailx* variables and aliases.
5. Process the start-up file of *mailx* commands named in the user *MAILRC* variable.

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

A blank line in a start-up file shall be ignored.

Internal Variables in mailx

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 **Commands in mailx** (on page 2795) concerning filenames and quoting shall also apply.

The defaults specified here may be changed by the implementation-defined system start-up file unless the user specifies the **–n** option.

allnet All network names whose login name components match shall be treated as identical. This shall cause the *msglist* message specifications to behave similarly. The default shall be **noallnet**. See also the **alternates** command and the **metoo** variable.

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 **noask** and **asksub**. The default shall be **asksub**, but no prompting shall be done if standard input is not a terminal.

askbcc Prompt for the blind copy list. The default shall be **noaskbcc**.

askcc Prompt for the copy 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 the **escape** command and **Command Escapes in mailx** (on page 2803). 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. 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.

XSI **debug** Enable verbose diagnostics for debugging. Messages are not delivered. The default shall be **nodebug**.

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 is 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. 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 added as a prefix to each line that is inserted into the message by the **~m** command escape. This variable shall default to one <tab>.

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 the messages that have been saved from the system mailbox into other files in the file designated by the variable *MBOX*, 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**.

XSI **onehop** When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (that is, one hop away). The default shall be **noonehop**.

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, the baud rate, or some combination of these shall be used.

sendwait Wait for the background mailer to finish before returning. The default shall be **nosendwait**.

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>s and <newline>s, respectively. (See also **~i** in **Command Escapes in mailx** (on page 2803).)

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>s and <newline>s, respectively.

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

Commands in mailx

The following *mailx* commands shall be provided. In the following list, header refers to lines from the message header, as shown in the OUTPUT FILES section. Header-line refers to lines within the header that begin with one or more non-white-space characters, immediately followed by a colon and white space and continuing until the next line beginning with a non-white-space character or an empty line. 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.

Filenames, where expected, shall be subjected to the process of shell word expansions (see Section 2.6 (on page 2238)); if more than a single pathname results and the command is expecting one file, the effects are unspecified. If the filename begins with an unquoted plus sign, it shall not be expanded, but treated as the named file (less the leading plus) in the **folder** directory. (See the **folder** variable.)

Declare Aliases

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

Add the given addresses to the alias specified by *alias*. The names shall be substituted when *alias* is used as a recipient address specified by the user in an outgoing message (that is, 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; for example, 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.

22864 **Declare Alternatives**

22865 *Synopsis*: `alt[ernates] name...`

22866 (See also the **metoo** command.) Declare a list of alternative names for the user's login. When
22867 responding to a message, these names shall be removed from the list of recipients for the
22868 response. The comparison of names shall be in a case-insensitive manner. With no arguments,
22869 **alternates** shall write the current list of alternative names.

22870 **Change Current Directory**

22871 *Synopsis*: `cd [directory]`
22872 `ch[dir] [directory]`

22873 Change directory. If *directory* is not specified, the contents of *HOME* shall be used.

22874 **Copy Messages**

22875 *Synopsis*: `c[opy] [file]`
22876 `c[opy] [msglist] file`
22877 `C[opy] [msglist]`

22878 Copy messages to the file named by the pathname *file* without marking the messages as saved.
22879 Otherwise, it shall be equivalent to the **save** command.

22880 In the capitalized form, save the specified messages in a file whose name is derived from the
22881 author of the message to be saved, without marking the messages as saved. Otherwise, it shall
22882 be equivalent to the **Save** command.

22883 **Delete Messages**

22884 *Synopsis*: `d[elete] [msglist]`

22885 Mark messages for deletion from the mailbox. The deletions shall not occur until *mailx* quits (see
22886 the **quit** command) or changes mailboxes (see the **folder** command). If **autoprint** is set and there
22887 are messages remaining after the **delete** command, the current message shall be written as
22888 described for the **print** command (see the **print** command); otherwise, the *mailx* prompt shall be
22889 written.

22890 **Discard Header Fields**

22891 *Synopsis*: `di[scard] [header-field...]`
22892 `ig[nore] [header-field...]`

22893 Suppress the specified header fields when writing messages. Specified *header-fields* shall be
22894 added to the list of suppressed header fields. Examples of header fields to ignore are **status** and
22895 **cc**. The fields shall be included when the message is saved. The **Print** and **Type** commands shall
22896 override this command. The comparison of header fields shall be in a case-insensitive manner. If
22897 no arguments are specified, write a list of the currently suppressed header fields to standard
22898 output; the listing need not reflect the same order of header fields that were entered.

22899 If both **retain** and **discard** commands are given, **discard** commands shall be ignored.

22900 **Delete Messages and Display**

22901 *Synopsis*: `dp [`*`msglist`*`]`
22902 `dt [`*`msglist`*`]`

22903 Delete the specified messages as described for the **delete** command, except that the **autoprint**
22904 variable shall have no effect, and the current message shall be written only if it was set to a
22905 message after the last message deleted by the command. Otherwise, an informational message
22906 to the effect that there are no further messages in the mailbox shall be written, followed by the
22907 *mailx* prompt.

22908 **Echo a String**

22909 *Synopsis*: `ec[ho]` *`string ...`*

22910 Echo the given strings, equivalent to the shell *echo* utility.

22911 **Edit Messages**

22912 *Synopsis*: `e[dit] [`*`msglist`*`]`

22913 Edit the given messages. The messages shall be placed in a temporary file and the utility named
22914 by the *EDITOR* variable is invoked to edit each file in sequence. The default *EDITOR* is
22915 unspecified.

22916 The **edit** command does not modify the contents of those messages in the mailbox.

22917 **Exit**

22918 *Synopsis*: `ex[it]`
22919 `x[it]`

22920 Exit from *mailx* without changing the mailbox. No messages shall be saved in the **mbox** (see also
22921 **quit**).

22922 **Change Folder**

22923 *Synopsis*: `fi[le] [`*`file`*`]`
22924 `fold[er] [`*`file`*`]`

22925 Quit (see the **quit** command) from the current file of messages and read in the file named by the
22926 pathname *file*. If no argument is given, the name and status of the current mailbox shall be
22927 written.

22928 Several unquoted special characters shall be recognized when used as *file* names, with the
22929 following substitutions:

22930 `%` The system mailbox for the invoking user.

22931 `%`*`user`* The system mailbox for *user*.

22932 `#` The previous file.

22933 `&` The current **mbox**.

22934 `+`*`file`* The named file in the **folder** directory. (See the **folder** variable.)

22935 The default file shall be the current mailbox.

22936 **Display List of Folders**

22937 *Synopsis*: `folders`

22938 Write the names of the files in the directory set by the **folder** variable. The command specified by
22939 the *LISTER* environment variable shall be used (see the ENVIRONMENT VARIABLES section).

22940 **Follow Up Specified Messages**

22941 *Synopsis*: `fo[llowup] [`*`message`*`]`
22942 `F[ollowup] [`*`msglist`*`]`

22943 In the lowercase form, respond to a message, recording the response in a file whose name is
22944 derived from the author of the message. See also the **save** and **copy** commands and **outfolder**.

22945 In the capitalized form, respond to the first message in the *msglist*, sending the message to the
22946 author of each message in the *msglist*. The subject line shall be taken from the first message and
22947 the response shall be recorded in a file whose name is derived from the author of the first
22948 message. See also the **Save** and **Copy** commands and **outfolder**.

22949 Both forms shall override the **record** variable, if set.

22950 **Display Header Summary for Specified Messages**

22951 *Synopsis*: `f[rom] [`*`msglist`*`]`

22952 Write the header summary for the specified messages.

22953 **Display Header Summary**

22954 *Synopsis*: `h[eaders] [`*`message`*`]`

22955 Write the page of headers that includes the message specified. If the *message* argument is not
22956 specified, the current message shall not change. However, if the *message* argument is specified,
22957 the current message shall become the message that appears at the top of the page of headers that
22958 includes the message specified. The **screen** variable sets the number of headers per page. See
22959 also the **z** command.

22960 **Help**

22961 *Synopsis*: `hel[p]`
22962 `?`

22963 Write a summary of commands.

22964 **Hold Messages**

22965 *Synopsis*: `ho[ld] [`*`msglist`*`]`
22966 `pre[serve] [`*`msglist`*`]`

22967 Mark the messages in *msglist* to be retained in the mailbox when *mailx* terminates. This shall
22968 override any commands that might previously have marked the messages to be deleted. During
22969 the current invocation of *mailx*, only the **delete**, **dp**, or **dt** commands shall remove the *preserve*
22970 marking of a message.

22971 **Execute Commands Conditionally**

22972 *Synopsis*: `i[f] s|r`
22973 *`mail-command`*`s`
22974 `el[se]`
22975 *`mail-command`*`s`
22976 `en[dif]`

22977 Execute commands conditionally, where **if s** executes the following *mail-command*s, up to an
22978 **else** or **endif**, if the program is in Send Mode, and **if r** shall cause the *mail-command*s to be
22979 executed only in Receive Mode.

22980 **List Available Commands**

22981 *Synopsis*: `l[ist]`

22982 Write a list of all commands available. No explanation shall be given.

22983 **Mail a Message**

22984 *Synopsis*: `m[ail]` *`address...`*

22985 Mail a message to the specified addresses or aliases.

22986 **Direct Messages to mbox**

22987 *Synopsis*: `mb[ox] [`*`msglist`*`]`

22988 Arrange for the given messages to end up in the **mbox** save file when *mailx* terminates normally.
22989 See *MBOX*. See also the **exit** and **quit** commands.

22990 **Process Next Specified Message**

22991 *Synopsis*: `n[ext] [`*`message`*`]`

22992 If the current message has not been written (for example, by the **print** command) since *mailx*
22993 started or since any other message was the current message, behave as if the **print** command
22994 was entered. Otherwise, if there is an undeleted message after the current message, make it the
22995 current message and behave as if the **print** command was entered. Otherwise, an informational
22996 message to the effect that there are no further messages in the mailbox shall be written, followed
22997 by the *mailx* prompt.

22998 **Pipe Message**

22999 *Synopsis*: `pi[pe] [[`*`msglist`*`] `*`command`*`]`
23000 `| [[`*`msglist`*`] `*`command`*`]`

23001 Pipe the messages through the given *command* by invoking the command interpreter specified
23002 by *SHELL* with two arguments: **–c** and *command*. (See also *sh* **–c**.) The application shall ensure
23003 that the command is given as a single argument. Quoting, described previously, can be used to
23004 accomplish this. If no arguments are given, the current message shall be piped through the
23005 command specified by the value of the **cmd** variable. If the **page** variable is set, a <form-feed>
23006 shall be inserted after each message.

23007 **Display Message with Headers**

23008 *Synopsis*: `P[rint] [`*`msglist`*`]`
23009 `T[ype] [`*`msglist`*`]`

23010 Write the specified messages, including all header lines, to standard output. Override
23011 suppression of lines by the **discard**, **ignore**, and **retain** commands. If **crt** is set, the messages
23012 longer than the number of lines specified by the **crt** variable shall be paged through the
23013 command specified by the *PAGER* environment variable.

23014 **Display Message**

23015 *Synopsis*: `p[rint] [`*`msglist`*`]`
23016 `t[ype] [`*`msglist`*`]`

23017 Write the specified messages to standard output. If **crt** is set, the messages longer than the
23018 number of lines specified by the **crt** variable shall be paged through the command specified by
23019 the *PAGER* environment variable.

23020 **Quit**

23021 *Synopsis*: `q[uit]`
23022 *`end-of-file`*

23023 Terminate *mailx*, storing messages that were read in **mbox** (if the current mailbox is the system
23024 mailbox and unless **hold** is set), deleting messages that have been explicitly saved (unless
23025 **keepsave** is set), discarding messages that have been deleted, and saving all remaining messages
23026 in the mailbox.

23027 **Reply to a Message List**

23028 *Synopsis*: `R[eply] [`*`msglist`*`]`
23029 `R[espond] [`*`msglist`*`]`

23030 Mail a reply message to the sender of each message in the *msglist*. The subject line shall be
23031 formed by concatenating **Re:**<space> (unless it already begins with that string) and the subject
23032 from the first message. If **record** is set to a filename, the response shall be saved at the end of that
23033 file.

23034 See also the **flipr** variable.

23035 **Reply to a Message**

23036 *Synopsis*: `r[eply] [`*`message`*`]`
23037 `r[espond] [`*`message`*`]`

23038 Mail a reply message to all recipients included in the header of the message. The subject line
23039 shall be formed by concatenating **Re:**<space> (unless it already begins with that string) and the
23040 subject from the message. If **record** is set to a filename, the response shall be saved at the end of
23041 that file.

23042 See also the **flipr** variable.

23043 **Retain Header Fields**

23044 *Synopsis*: `ret[ain] [`*`header-field`*`...]`

23045 Retain the specified header fields when writing messages. This command shall override all
23046 **discard** and **ignore** commands. The comparison of header fields shall be in a case-insensitive
23047 manner. If no arguments are specified, write a list of the currently retained header fields to
23048 standard output; the listing need not reflect the same order of header fields that were entered.

23049 **Save Messages**

23050 *Synopsis*: `s[ave] [`*`file`*`]`
23051 `s[ave] [`*`msglist`*`] `*`file`*
23052 `S[ave] [`*`msglist`*`]`

23053 Save the specified messages in the file named by the pathname *file*, or the **mbox** if the *file*
23054 argument is omitted. The file shall be created if it does not exist; otherwise, the messages shall be
23055 appended to the file. The message shall be put in the state *saved*, and shall behave as specified in
23056 the description of the *saved* state when the current mailbox is exited by the **quit** or **file**
23057 command.

23058 In the capitalized form, save the specified messages in a file whose name is derived from the
23059 author of the first message. The name of the file shall be taken to be the author's name with all
23060 network addressing stripped off. See also the **Copy**, **followup**, and **Followup** commands and
23061 **outfolder** variable.

23062 **Set Variables**

23063 *Synopsis*: `se[t] [`*`name`*`[=[`*`string`*`]] ...] [`*`name=number ...`*`] [no`*`name ...`*`]`

23064 Define one or more variables called *name*. The variable can be given a null, string, or numeric
23065 value. Quoting and backslash escapes can occur anywhere in *string*, as described previously, as
23066 if the *string* portion of the argument were the entire argument. The forms *name* and *name*= shall
23067 be equivalent to *name*="" for variables that take string values. The **set** command without
23068 arguments shall write a list of all defined variables and their values. The **no** *name* form shall be
23069 equivalent to **unset** *name*.

23070 **Invoke a Shell**

23071 *Synopsis*: `sh[ell]`

23072 Invoke an interactive command interpreter (see also *SHELL*).

23073 **Display Message Size**

23074 *Synopsis*: `si[ze] [`*`msglist`*`]`

23075 Write the size in bytes of each of the specified messages.

23076 **Read mailx Commands From a File**

23077 *Synopsis*: `so[urce] `*`file`*

23078 Read and execute commands from the file named by the pathname *file* and return to command
23079 mode.

23080 **Display Beginning of Messages**

23081 *Synopsis*: `to[p] [`*`msglist`*`]`

23082 Write the top few lines of each of the specified messages. If the **toplines** variable is set, it is taken
23083 as the number of lines to write. The default shall be 5.

23084 **Touch Messages**

23085 *Synopsis*: `tou[ch] [`*`msglist`*`]`

23086 Touch the specified messages. If any message in *msglist* is not specifically deleted nor saved in a
23087 file, it shall be placed in the **mbox** upon normal termination. See **exit** and **quit**.

23088 **Delete Aliases**

23089 *Synopsis*: `una[lias] [`*`alias`*`]...`

23090 Delete the specified alias names. If a specified alias does not exist, the results are unspecified.

23091 **Undelete Messages**

23092 *Synopsis*: `u[ndelete] [`*`msglist`*`]`

23093 Change the state of the specified messages from deleted to read. If **autoprint** is set, the last
23094 message of those restored shall be written. If *msglist* is not specified, the message shall be
23095 selected as follows:

23096 • If there are any deleted messages that follow the current message, the first of these shall be
23097 chosen.

23098 • Otherwise, the last deleted message that also precedes the current message shall be chosen.

23099 **Unset Variables**

23100 *Synopsis*: `uns[et]` *`name`*`...`

23101 Cause the specified variables to be erased.

23102 **Edit Message with Full-Screen Editor**

23103 *Synopsis*: `v[isual] [`*`msglist`*`]`

23104 Edit the given messages with a screen editor. Each message shall be placed in a temporary file,
23105 and the utility named by the *VISUAL* variable shall be invoked to edit each file in sequence. The
23106 default editor shall be *vi*.

23107 The **visual** command does not modify the contents of those messages in the mailbox.

23108 **Write Messages to a File**

23109 *Synopsis*: `w[rite] [`*`msglist`*`]` *`file`*

23110 Write the given messages to the file specified by the pathname *file*, minus the message header.
23111 Otherwise, it shall be equivalent to the **save** command.

23112 **Scroll Header Display**

23113 *Synopsis*: `z[+|−]`

23114 Scroll the header display forward (if `'+'` is specified or if no option is specified) or backward (if
23115 `'−'` is specified) one screenful. The number of headers written shall be set by the **screen**
23116 variable.

23117 **Invoke Shell Command**

23118 *Synopsis*: `!command`

23119 Invoke the command interpreter specified by *SHELL* with two arguments: **−c** and *command*.
23120 (See also *sh* **−c**.) If the **bang** variable is set, each unescaped occurrence of `'!'` in command shall
23121 be replaced with the command executed by the previous **!** command or **~!** command escape.

23122 **Null Command**

23123 *Synopsis*: `# comment`

23124 This null command (comment) shall be ignored by *mailx*.

23125 **Display Current Message Number**

23126 *Synopsis*: `=`

23127 Write the current message number.

23128 **Command Escapes in mailx**

23129 The following commands can be entered only from input mode, by beginning a line with the
23130 escape character (by default, tilde (`'~'`)). See the **escape** variable description for changing this
23131 special character. The format for the commands shall be:

23132 `<escape-character><command-char><separator>[<arguments>]`

23133 where the <*separator*> can be zero or more <blank>s.

23134 In the following descriptions, the application shall ensure that the argument *command* (but not
23135 *mailx-command)* is a shell command string. Any string acceptable to the command interpreter
23136 specified by the *SHELL* variable when it is invoked as *SHELL* **−c** *command_string* shall be valid.
23137 The command can be presented as multiple arguments (that is, quoting is not required).

23138 Command escapes that are listed with *msglist* or *mailx-command* arguments are invalid in Send
23139 Mode and produce unspecified results.

23140 **~!** *command* Invoke the command interpreter specified by *SHELL* with two arguments: **−c** and
23141 *command*; and then return to input mode. If the **bang** variable is set, each
23142 unescaped occurrence of `'!'` in *command* shall be replaced with the command
23143 executed by the previous **!** command or **~!** command escape.

23144 **~.** Simulate end-of-file (terminate message input).

23145 **~:** *mailx-command*, **~_** *mailx-command*
23146 Perform the command-level request.

23147 **~?** Write a summary of command escapes.

23148 **~A** This shall be equivalent to **~i Sign**.

23149 **~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* for a description of this file.

~e Invoke the editor, as specified by the *EDITOR* environment variable, 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 also shall 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.

~m [*msglist***]** Insert the specified messages into the message, prefixing non-empty lines with the string in the **indentprefix** variable. This command escape also shall 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 than **crt** lines (see **Internal Variables in mailx** (on page 2792)), the output shall be paginated as described for the *PAGER* variable.

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

"**~r** *file*, **~<** *file*, **~r** **!***command*, **~<** **!***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 command interpreter specified by *SHELL* shall be invoked with two arguments: **–c** and *command*. The standard output of *command* shall be inserted into the message.

~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, 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 invoking the command interpreter specified by *SHELL* with two arguments: **–c** and *command*. If the *command* returns a successful exit status, the standard output 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.

EXIT STATUS

When the **–e** option is specified, the following exit values are returned:

0 Mail was found.

>0 Mail was not found or an error occurred.

Otherwise, the following exit values are returned:

0 Successful completion; note that this status implies that all messages were *sent*, but it gives no assurances that any of them were actually *delivered*.

>0 An error occurred.

CONSEQUENCES OF ERRORS

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

- If an error is encountered processing a command escape (see **Command Escapes in mailx** (on page 2803)), a diagnostic message shall be written to standard error, and the message being composed may be modified, but this condition shall not prevent the message from being sent.
- Other errors shall prevent the sending of the message.

When in command mode:

- Default.

APPLICATION USAGE

Delivery of messages to remote systems requires the existence of communication paths to such systems. These need not exist.

Input lines are limited to {LINE_MAX} bytes, but mailers between systems may impose more severe line-length restrictions. This volume of IEEE Std 1003.1-200x 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 100,000 bytes because some mail gateways impose message-length restrictions.

The format of the system mailbox is intentionally unspecified. Not all systems 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 are not 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.

The **discard** and **ignore** commands are not inverses of the **retain** command. The **retain** command discards all header-fields except those explicitly retained. The **discard** command keeps all header-fields except those explicitly discarded. If headers exist on the retained header list, **discard** and **ignore** commands are ignored.

EXAMPLES

None.

RATIONALE

The standard developers felt strongly that a method for applications to send messages to specific users was necessary. The obvious example is a batch utility, running non-interactively, 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 volume of IEEE Std 1003.1-200x.

The intent of this command is to provide a simple, portable interface for sending messages non-interactively. 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 described 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.

Many historical systems modified each body line that started with **From** by prefixing the `'F'` with `'>'`. It is unnecessary, but allowed, to do that when the string does not follow a blank line because it cannot be confused with the next header.

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 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 are not able to handle lines that long, however.

Like the utilities *logger* and *lp*, *mailx* admittedly is difficult to test. This was not deemed sufficient justification to exclude this utility from this volume of IEEE Std 1003.1-200x. It is also arguable that it is, 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 *mailx* ASYNCHRONOUS EVENTS. 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 timezone in which the mail originated, and, therefore, *mailx* cannot convert the date and time strings into the internal form that then is 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.

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

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

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 **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 in this volume of IEEE Std 1003.1-200x.

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 ampersand. Since this volume of IEEE Std 1003.1-200x chose the *mailx* name, it kept the System V default, assuming that BSD users would not have difficulty with this minor incompatibility (that they can override).

The meanings of **r** and **R** are reversed between System V *mailx* and SunOS *Mail*. Once again, since this volume of IEEE Std 1003.1-200x chose the *mailx* name, it kept the System V default, but allows the SunOS user to achieve the desired results using **flipr**, an internal variable in System V *mailx*, although it has not been documented in the SVID

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

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 start-up file'' historically has been named **/etc/mailx.rc**, but this specific name and location are not required.

The intent of the wording for the **next** command is that if any command has already displayed the current message it should display a following message, but, otherwise, it should display the current message. Consider the command sequence:

```
next 3
delete 3
```

23329 `next`

23330 where the **autoprint** option was not set. The normative text specifies that the second **next**
23331 command should display a message following the third message, because even though the
23332 current message has not been displayed since it was set by the **delete** command, it has been
23333 displayed since the current message was anything other than message number 3. This does not
23334 always match historical practice in some implementations, where the command file address
23335 followed by **next** (or the default command) would skip the message for which the user had
23336 searched.

23337 **FUTURE DIRECTIONS**

23338 None.

23339 **SEE ALSO**

23340 *ed*, *ls*, *more*, *vi*

23341 **CHANGE HISTORY**

23342 First released in Issue 2.

23343 **Issue 5**

23344 The description of the EDITOR environment variable is changed to indicate that *ed* is the default
23345 editor if this variable is not set. In previous issues, this default was not stated explicitly at this
23346 point but was implied further down in the text.

23347 FUTURE DIRECTIONS section added.

23348 **Issue 6**

23349 The following new requirements on POSIX implementations derive from alignment with the
23350 Single UNIX Specification:

23351 - The **–F** option is added.

23352 - The **allnet**, **debug**, and **sendwait** internal variables are added.

23353 - The **C**, **ec**, **fo**, **F**, and **S** *mailx* commands are added.

23354 In the DESCRIPTION and ENVIRONMENT VARIABLES sections, text stating ‘‘*HOME*
23355 directory’’ is replaced by ‘‘directory referred to by the *HOME* environment variable’’.

23356 The *mailx* utility is aligned with the IEEE P1003.2b draft standard, which included various
23357 clarifications to resolve IEEE PASC Interpretations submitted for the ISO POSIX-2: 1993
23358 standard. In particular, the changes here address IEEE PASC Interpretations 1003.2 #10, #11,
23359 #103, #106, #108, #114, #115, #122, and #129.

23360 The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

23361 The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

NAME

make — maintain, update, and regenerate groups of programs (**DEVELOPMENT**)

SYNOPSIS

SD
```
make [–einpqrst][–f makefile]...[ –k| –S][macro=value]...
    [target_name...]
```

DESCRIPTION

The *make* utility shall update files that are derived from other files. A typical case is one where object files are derived from the corresponding source files. The *make* utility examines time relationships and shall update those derived files (called targets) that have modified times earlier than the modified times of the files (called prerequisites) from which they are derived. A description file (makefile) contains a description of the relationships between files, and the commands that need to be executed to update the targets to reflect changes in their prerequisites. Each specification, or rule, shall consist of a target, optional prerequisites, and optional commands to be executed when a prerequisite is newer than the target. There are two types of rule:

1. *Inference rules*, which have one target name with at least one period (`'.'`) and no slash (`'/'`)
2. *Target rules*, which can have more than one target name

In addition, *make* shall have a collection of built-in macros and inference rules that infer prerequisite relationships to simplify maintenance of programs.

To receive exactly the behavior described in this section, the user shall ensure that a portable makefile shall:

- Include the special target **.POSIX**
- Omit any special target reserved for implementations (a leading period followed by uppercase letters) that has not been specified by this section

The behavior of *make* is unspecified if either or both of these conditions are not met.

OPTIONS

The *make* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–e Cause environment variables, including those with null values, to override macro assignments within makefiles.

–f *makefile* Specify a different makefile. The argument *makefile* is a pathname of a description file, which is also referred to as the *makefile*. A pathname of `'–'` shall denote the standard input. There can be multiple instances of this option, and they shall be processed in the order specified. The effect of specifying the same option-argument more than once is unspecified.

–i Ignore error codes returned by invoked commands. This mode is the same as if the special target **.IGNORE** were specified without prerequisites.

–k Continue to update other targets that do not depend on the current target if a non-ignored error occurs while executing the commands to bring a target up-to-date.

–n Write commands that would be executed on standard output, but do not execute them. However, lines with a plus sign (`'+'`) prefix shall be executed. In this mode,

lines with an at sign (`'@'`) character prefix shall be written to standard output.

–p Write to standard output the complete set of macro definitions and target descriptions. The output format is unspecified.

–q Return a zero exit value if the target file is up-to-date; otherwise, return an exit value of 1. Targets shall not be updated if this option is specified. However, a makefile command line (associated with the targets) with a plus sign (`'+'`) prefix shall be executed.

–r Clear the suffix list and does not use the built-in rules.

–S Terminate *make* if an error occurs while executing the commands to bring a target up-to-date. This shall be the default and the opposite of **–k**.

–s Do not write makefile command lines or touch messages (see **–t**) to standard output before executing. This mode shall be the same as if the special target **.SILENT** were specified without prerequisites.

–t Update the modification time of each target as though a *touch target* had been executed. Targets that have prerequisites but no commands (see **Target Rules** (on page 2813)), or that are already up-to-date, shall not be touched in this manner. Write messages to standard output for each target file indicating the name of the file and that it was touched. Normally, the makefile command lines associated with each target are not executed. However, a command line with a plus sign (`'+'`) prefix shall be executed.

Any options specified in the *MAKEFLAGS* environment variable shall be evaluated before any options specified on the *make* utility command line. If the **–k** and **–S** options are both specified on the *make* utility command line or by the *MAKEFLAGS* environment variable, the last option specified shall take precedence. If the **–f** or **–p** options appear in the *MAKEFLAGS* environment variable, the result is undefined.

OPERANDS

The following operands shall be supported:

target_name Target names, as defined in the EXTENDED DESCRIPTION section. If no target is specified, while *make* is processing the makefiles, the first target that *make* encounters that is not a special target or an inference rule shall be used.

macro=value Macro definitions, as defined in **Macros** (on page 2815).

If the *target_name* and *macro=value* operands are intermixed on the *make* utility command line, the results are unspecified.

STDIN

The standard input shall be used only if the *makefile* option-argument is `'–'`. See the INPUT FILES section.

INPUT FILES

The input file, otherwise known as the makefile, is a text file containing rules, macro definitions, and comments. See the EXTENDED DESCRIPTION section.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *make*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

MAKEFLAGS
This variable shall be interpreted as a character string representing a series of option characters to be used as the default options. The implementation shall accept both of the following formats (but need not accept them when intermixed):

- The characters are option letters without the leading hyphens or <blank> separation used on a *make* utility command line.
- The characters are formatted in a manner similar to a portion of the *make* utility command line: options are preceded by hyphens and <blank>-separated as described in the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines. The *macro=value* macro definition operands can also be included. The difference between the contents of *MAKEFLAGS* and the *make* utility command line is that the contents of the variable shall not be subjected to the word expansions (see Section 2.6 (on page 2238)) associated with parsing the command line values.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

XSI *PROJECTDIR*
Provide a directory to be used to search for SCCS files not found in the current directory. In all of the following cases, the search for SCCS files is made in the directory **SCCS** in the identified directory. If the value of *PROJECTDIR* begins with a slash, it shall be considered an absolute pathname; otherwise, the value of *PROJECTDIR* is treated as a user name and that user's initial working directory shall be examined for a subdirectory **src** or **source**. If such a directory is found, it shall be used. Otherwise, the value is used as a relative pathname.

If *PROJECTDIR* is not set or has a null value, the search for SCCS files shall be made in the directory **SCCS** in the current directory.

The setting of *PROJECTDIR* affects all files listed in the remainder of this utility description for files with a component named **SCCS**.

The value of the *SHELL* environment variable shall not be used as a macro and shall not be modified by defining the **SHELL** macro in a makefile or on the command line. All other environment variables, including those with null values, shall be used as macros, as defined in **Macros** (on page 2815).

ASYNCHRONOUS EVENTS

If not already ignored, *make* shall trap SIGHUP, SIGTERM, SIGINT, and SIGQUIT and remove the current target unless the target is a directory or the target is a prerequisite of the special target **.PRECIOUS** or unless one of the **–n**, **–p**, or **–q** options was specified. Any targets removed in this manner shall be reported in diagnostic messages of unspecified format, written to standard error. After this cleanup process, if any, *make* shall take the standard action for all other signals.

23497 **STDOUT**

23498 The *make* utility shall write all commands to be executed to standard output unless the –**s** option
23499 was specified, the command is prefixed with an at sign, or the special target **.SILENT** has either
23500 the current target as a prerequisite or has no prerequisites. If *make* is invoked without any work
23501 needing to be done, it shall write a message to standard output indicating that no action was
23502 taken. If the –**t** option is present and a file is touched, *make* shall write to standard output a
23503 message of unspecified format indicating that the file was touched, including the filename of the
23504 file.

23505 **STDERR**

23506 The standard error shall be used only for diagnostic messages.

23507 **OUTPUT FILES**

23508 Files can be created when the –**t** option is present. Additional files can also be created by the
23509 utilities invoked by *make*.

23510 **EXTENDED DESCRIPTION**

23511 The *make* utility attempts to perform the actions required to ensure that the specified targets are
23512 up-to-date. A target is considered out-of-date if it is older than any of its prerequisites or if it
23513 does not exist. The *make* utility shall treat all prerequisites as targets themselves and recursively
23514 ensure that they are up-to-date, processing them in the order in which they appear in the rule.
23515 The *make* utility shall use the modification times of files to determine whether the corresponding
23516 targets are out-of-date.

23517 After *make* has ensured that all of the prerequisites of a target are up-to-date and if the target is
23518 out-of-date, the commands associated with the target entry shall be executed. If there are no
23519 commands listed for the target, the target shall be treated as up-to-date.

23520 **Makefile Syntax**

23521 A makefile can contain rules, macro definitions (see **Macros** (on page 2815)), and comments.
23522 There are two kinds of rules: *inference rules* and *target rules*. The *make* utility shall contain a set of
23523 built-in inference rules. If the –**r** option is present, the built-in rules shall not be used and the
23524 suffix list shall be cleared. Additional rules of both types can be specified in a makefile. If a rule
23525 is defined more than once, the value of the rule shall be that of the last one specified. Macros can
23526 also be defined more than once, and the value of the macro is specified in **Macros** (on page
23527 2815). Comments start with a number sign (′#′) and continue until an unescaped <newline> is
23528 reached.

23529 By default, the following files shall be tried in sequence: **./makefile** and **./Makefile**. If neither
23530 XSI **./makefile** or **./Makefile** are found, other implementation-defined files may also be tried. On
23531 XSI-conformant systems, the additional files **./s.makefile**, **SCCS/s.makefile**, **./s.Makefile**, and
23532 **SCCS/s.Makefile** shall also be tried.

23533 The –**f** option shall direct *make* to ignore any of these default files and use the specified argument
23534 as a makefile instead. If the ′–′ argument is specified, standard input shall be used.

23535 The term *makefile* is used to refer to any rules provided by the user, whether in **./makefile** or its
23536 variants, or specified by the –**f** option.

23537 The rules in makefiles shall consist of the following types of lines: target rules, including special
23538 targets (see **Target Rules** (on page 2813)), inference rules (see **Inference Rules** (on page 2816)),
23539 macro definitions (see **Macros** (on page 2815)), empty lines, and comments.

23540 When an escaped <newline> (one preceded by a backslash) is found anywhere in the makefile
23541 except in a command line, it shall be replaced, along with any leading white space on the
23542 following line, with a single <space>. When an escaped <newline> is found in a command line

in a makefile, the command line shall contain the backslash, the <newline>, and the next line, except that the first character of the next line shall not be included if it is a <tab>.

Makefile Execution

Makefile command lines shall be processed one at a time by writing the makefile command line to the standard output (unless one of the conditions listed under '@' suppresses the writing) and executing the command(s) in the line. A <tab> may precede the command to standard output. Command execution shall be as if the makefile command line were the argument to the *system*( ) function. The environment for the command being executed shall contain all of the variables in the environment of *make*.

By default, when *make* receives a non-zero status from the execution of a command, it shall terminate with an error message to standard error.

Makefile command lines can have one or more of the following prefixes: a hyphen ('–'), an at sign ('@'), or a plus sign ('+'). These shall modify the way in which *make* processes the command. When a command is written to standard output, the prefix shall not be included in the output.

– If the command prefix contains a hyphen, or the **–i** option is present, or the special target **.IGNORE** has either the current target as a prerequisite or has no prerequisites, any error found while executing the command shall be ignored.

@ If the command prefix contains an at sign and the *make* utility command line **–n** option is not specified, or the **–s** option is present, or the special target **.SILENT** has either the current target as a prerequisite or has no prerequisites, the command shall not be written to standard output before it is executed.

\+ If the command prefix contains a plus sign, this indicates a makefile command line that shall be executed even if **–n**, **–q**, or **–t** is specified.

Target Rules

Target rules are formatted as follows:

```
target [target...]: [prerequisite...][;command]
[<tab>command
<tab>command
...]

line that does not begin with <tab>
```

Target entries are specified by a <blank>-separated, non-null list of targets, then a colon, then a <blank>-separated, possibly empty list of prerequisites. Text following a semicolon, if any, and all following lines that begin with a <tab>, are makefile command lines to be executed to update the target. The first non-empty line that does not begin with a <tab> or '#' shall begin a new entry. An empty or blank line, or a line beginning with '#', may begin a new entry.

Applications shall select target names from the set of characters consisting solely of periods, underscores, digits, and alphabetics from the portable character set (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set). Implementations may allow other characters in target names as extensions. The interpretation of targets containing the characters '%' and '"' is implementation-defined.

A target that has prerequisites, but does not have any commands, can be used to add to the prerequisite list for that target. Only one target rule for any given target can contain commands.

Lines that begin with one of the following are called *special targets* and control the operation of *make*:

.DEFAULT If the makefile uses this special target, the application shall ensure that it is specified with commands, but without prerequisites. The commands shall be used by *make* if there are no other rules available to build a target.

.IGNORE Prerequisites of this special target are targets themselves; this shall cause errors from commands associated with them to be ignored in the same manner as specified by the **–i** option. Subsequent occurrences of **.IGNORE** shall add to the list of targets ignoring command errors. If no prerequisites are specified, *make* shall behave as if the **–i** option had been specified and errors from all commands associated with all targets shall be ignored.

.POSIX The application shall ensure that this special target is specified without prerequisites or commands. If it appears as the first non-comment line in the makefile, *make* shall process the makefile as specified by this section; otherwise, the behavior of *make* is unspecified.

.PRECIOUS Prerequisites of this special target shall not be removed if *make* receives one of the asynchronous events explicitly described in the ASYNCHRONOUS EVENTS section. Subsequent occurrences of **.PRECIOUS** shall add to the list of precious files. If no prerequisites are specified, all targets in the makefile shall be treated as if specified with **.PRECIOUS**.

XSI **.SCCS_GET** The application shall ensure that this special target is specified without prerequisites. If this special target is included in a makefile, the commands specified with this target shall replace the default commands associated with this special target (see **Default Rules** (on page 2819)). The commands specified with this target are used to get all SCCS files that are not found in the current directory.

When source files are named in a dependency list, *make* shall treat them just like any other target. Because the source file is presumed to be present in the directory, there is no need to add an entry for it to the makefile. When a target has no dependencies, but is present in the directory, *make* shall assume that that file is up-to-date. If, however, an SCCS file named **SCCS/s.***source_file* is found for a target *source_file*, *make* compares the timestamp of the target file with that of the **SCCS/s.source_file** to assure the target is up-to-date. If the target is missing, or if the SCCS file is newer, *make* shall automatically issue the commands specified for the **.SCCS_GET** special target to retrieve the most recent version. However, if the target is writable by anyone, *make* shall not retrieve a new version.

.SILENT Prerequisites of this special target are targets themselves; this shall cause commands associated with them to not be written to the standard output before they are executed. Subsequent occurrences of **.SILENT** shall add to the list of targets with silent commands. If no prerequisites are specified, *make* shall behave as if the **–s** option had been specified and no commands or touch messages associated with any target shall be written to standard output.

.SUFFIXES Prerequisites of **.SUFFIXES** shall be appended to the list of known suffixes and are used in conjunction with the inference rules (see **Inference Rules** (on page 2816)). If **.SUFFIXES** does not have any prerequisites, the list of known suffixes shall be cleared.

The special targets **.IGNORE**, **.POSIX**, **.PRECIOUS**, **.SILENT**, and **.SUFFIXES** shall be specified without commands.

Targets with names consisting of a leading period followed by the uppercase letters "POSIX" and then any other characters are reserved for future standardization. Targets with names consisting of a leading period followed by one or more uppercase letters are reserved for implementation extensions.

Macros

Macro definitions are in the form:

string1 = **[***string2***]**

The macro named *string1* is defined as having the value of *string2*, where *string2* is defined as all characters, if any, after the equal sign, up to a comment character (*'#'*) or an unescaped <newline>. Any <blank>s immediately before or after the equal sign shall be ignored.

Applications shall select macro names from the set of characters consisting solely of periods, underscores, digits, and alphabetics from the portable character set (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set). A macro name shall not contain an equals sign. Implementations may allow other characters in macro names as extensions.

Macros can appear anywhere in the makefile. Macro expansions using the forms $(*string1*) or ${*string1*} shall be replaced by *string2*, as follows:

- Macros in target lines shall be evaluated when the target line is read.
- Macros in makefile command lines shall be evaluated when the command is executed.
- Macros in the string before the equals sign in a macro definition shall be evaluated when the macro assignment is made.
- Macros after the equals sign in a macro definition shall not be evaluated until the defined macro is used in a rule or command, or before the equals sign in a macro definition.

The parentheses or braces are optional if *string1* is a single character. The macro $$ shall be replaced by the single character *'$'*. If *string1* in a macro expansion contains a macro expansion, the results are unspecified.

Macro expansions using the forms $(*string1***[:***subst1***=[***subst2***]]**) or ${*string1***[:***subst1***=[***subst2***]]**} can be used to replace all occurrences of *subst1* with *subst2* when the macro substitution is performed. The *subst1* to be replaced shall be recognized when it is a suffix at the end of a word in *string1* (where a *word*, in this context, is defined to be a string delimited by the beginning of the line, a <blank> or <newline>). If *string1* in a macro expansion contains a macro expansion, the results are unspecified.

Macro expansions in *string1* of macro definition lines shall be evaluated when read. Macro expansions in *string2* of macro definition lines shall be performed when the macro identified by *string1* is expanded in a rule or command.

Macro definitions shall be taken from the following sources, in the following logical order, before the makefile(s) are read.

1. Macros specified on the *make* utility command line, in the order specified on the command line. It is unspecified whether the internal macros defined in **Internal Macros** (on page 2818) are accepted from this source.
2. Macros defined by the *MAKEFLAGS* environment variable, in the order specified in the environment variable. It is unspecified whether the internal macros defined in **Internal Macros** (on page 2818) are accepted from this source.

3. The contents of the environment, excluding the *MAKEFLAGS* and *SHELL* variables and including the variables with null values.

4. Macros defined in the inference rules built into *make*.

Macro definitions from these sources shall not override macro definitions from a lower-numbered source. Macro definitions from a single source (for example, the *make* utility command line, the *MAKEFLAGS* environment variable, or the other environment variables) shall override previous macro definitions from the same source.

Macros defined in the makefile(s) shall override macro definitions that occur before them in the makefile(s) and macro definitions from source 4. If the **–e** option is not specified, macros defined in the makefile(s) shall override macro definitions from source 3. Macros defined in the makefile(s) shall not override macro definitions from source 1 or source 2.

Before the makefile(s) are read, all of the *make* utility command line options (except **–f** and **–p**) and *make* utility command line macro definitions (except any for the *MAKEFLAGS* macro), not already included in the *MAKEFLAGS* macro, shall be added to the *MAKEFLAGS* macro, quoted in an implementation-defined manner such that when *MAKEFLAGS* is read by another instance of the *make* command, the original macro's value is recovered. Other implementation-defined options and macros may also be added to the *MAKEFLAGS* macro. If this modifies the value of the *MAKEFLAGS* macro, or, if the *MAKEFLAGS* macro is modified at any subsequent time, the *MAKEFLAGS* environment variable shall be modified to match the new value of the *MAKEFLAGS* macro. The result of setting *MAKEFLAGS* in the Makefile is unspecified.

Before the makefile(s) are read, all of the *make* utility command line macro definitions (except the *MAKEFLAGS* macro or the *SHELL* macro) shall be added to the environment of *make*. Other implementation-defined variables may also be added to the environment of *make*.

The **SHELL** macro shall be treated specially. It shall be provided by *make* and set to the pathname of the shell command language interpreter (see *sh* (on page 3048)). The *SHELL* environment variable shall not affect the value of the **SHELL** macro. If **SHELL** is defined in the makefile or is specified on the command line, it shall replace the original value of the **SHELL** macro, but shall not affect the *SHELL* environment variable. Other effects of defining **SHELL** in the makefile or on the command line are implementation-defined.

Inference Rules

Inference rules are formatted as follows:

```
target:
<tab>command
[<tab>command]
...
```

line that does not begin with <tab> *or* #

The application shall ensure that the *target* portion is a valid target name (see **Target Rules** (on page 2813)) of the form **.s2** or **.s1.s2** (where **.s1** and **.s2** are suffixes that have been given as prerequisites of the **.SUFFIXES** special target and *s1* and *s2* do not contain any slashes or periods.) If there is only one period in the target, it is a single-suffix inference rule. Targets with two periods are double-suffix inference rules. Inference rules can have only one target before the colon.

The application shall ensure that the makefile does not specify prerequisites for inference rules; no characters other than white space shall follow the colon in the first line, except when creating the *empty rule*, described below. Prerequisites are inferred, as described below.

Inference rules can be redefined. A target that matches an existing inference rule shall overwrite the old inference rule. An empty rule can be created with a command consisting of simply a semicolon (that is, the rule still exists and is found during inference rule search, but since it is empty, execution has no effect). The empty rule also can be formatted as follows:

```
rule: ;
```

where zero or more <blank>s separate the colon and semicolon.

The *make* utility uses the suffixes of targets and their prerequisites to infer how a target can be made up-to-date. A list of inference rules defines the commands to be executed. By default, *make* contains a built-in set of inference rules. Additional rules can be specified in the makefile.

The special target **.SUFFIXES** contains as its prerequisites a list of suffixes that shall be used by the inference rules. The order in which the suffixes are specified defines the order in which the inference rules for the suffixes are used. New suffixes shall be appended to the current list by specifying a **.SUFFIXES** special target in the makefile. A **.SUFFIXES** target with no prerequisites shall clear the list of suffixes. An empty **.SUFFIXES** target followed by a new **.SUFFIXES** list is required to change the order of the suffixes.

Normally, the user would provide an inference rule for each suffix. The inference rule to update a target with a suffix **.s1** from a prerequisite with a suffix **.s2** is specified as a target **.s2.s1**. The internal macros provide the means to specify general inference rules (see **Internal Macros** (on page 2818)).

When no target rule is found to update a target, the inference rules shall be checked. The suffix of the target (**.s1**) to be built is compared to the list of suffixes specified by the **.SUFFIXES** special targets. If the **.s1** suffix is found in **.SUFFIXES**, the inference rules shall be searched in the order defined for the first **.s2.s1** rule whose prerequisite file (**$*.s2**) exists. If the target is out-of-date with respect to this prerequisite, the commands for that inference rule shall be executed.

If the target to be built does not contain a suffix and there is no rule for the target, the single suffix inference rules shall be checked. The single-suffix inference rules define how to build a target if a file is found with a name that matches the target name with one of the single suffixes appended. A rule with one suffix **.s2** is the definition of how to build *target* from **target.s2**. The other suffix (**.s1**) is treated as null.

XSI A tilde (`'~'`) in the above rules refers to an SCCS file in the current directory. Thus, the rule **.c~.o** would transform an SCCS C-language source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix, it is incompatible with *make*'s suffix point of view. Hence, the `'~'` is a way of changing any file reference into an SCCS file reference.

Libraries

If a target or prerequisite contains parentheses, it shall be treated as a member of an archive library. For the *lib*(*member***.o**) expression *lib* refers to the name of the archive library and *member***.o** to the member name. The application shall ensure that the member is an object file with the **.o** suffix. The modification time of the expression is the modification time for the member as kept in the archive library; see *ar* (on page 2336). The **.a** suffix shall refer to an archive library. The **.s2.a** rule shall be used to update a member in the library from a file with a suffix **.s2**.

23761 **Internal Macros**

23762 The *make* utility shall maintain five internal macros that can be used in target and inference rules.
23763 In order to clearly define the meaning of these macros, some clarification of the terms *target rule*,
23764 *inference rule*, *target*, and *prerequisite* is necessary.

23765 Target rules are specified by the user in a makefile for a particular target. Inference rules are
23766 user-specified or *make*-specified rules for a particular class of target name. Explicit prerequisites
23767 are those prerequisites specified in a makefile on target lines. Implicit prerequisites are those
23768 prerequisites that are generated when inference rules are used. Inference rules are applied to
23769 implicit prerequisites or to explicit prerequisites that do not have target rules defined for them in
23770 the makefile. Target rules are applied to targets specified in the makefile.

23771 Before any target in the makefile is updated, each of its prerequisites (both explicit and implicit)
23772 shall be updated. This shall be accomplished by recursively processing each prerequisite. Upon
23773 recursion, each prerequisite shall become a target itself. Its prerequisites in turn shall be
23774 processed recursively until a target is found that has no prerequisites, at which point the
23775 recursion stops. The recursion then shall back up, updating each target as it goes.

23776 In the definitions that follow, the word *target* refers to one of:

23777 - A target specified in the makefile
23778 - An explicit prerequisite specified in the makefile that becomes the target when *make*
23779 processes it during recursion
23780 - An implicit prerequisite that becomes a target when *make* processes it during recursion

23781 In the definitions that follow, the word *prerequisite* refers to one of the following:

23782 - An explicit prerequisite specified in the makefile for a particular target
23783 - An implicit prerequisite generated as a result of locating an appropriate inference rule and
23784 corresponding file that matches the suffix of the target

23785 The five internal macros are:

23786 $@ The $@ shall evaluate to the full target name of the current target, or the archive
23787 filename part of a library archive target. It shall be evaluated for both target and
23788 inference rules.

23789 For example, in the **.c.a** inference rule, $@ represents the out-of-date **.a** file to be built.
23790 Similarly, in a makefile target rule to build **lib.a** from **file.c**, $@ represents the out-of-
23791 date **lib.a**.

23792 $% The $% macro shall be evaluated only when the current target is an archive library
23793 member of the form *libname*(*member***.o**). In these cases, $@ shall evaluate to *libname* and
23794 $% shall evaluates to *member***.o**. The $% macro shall be evaluated for both target and
23795 inference rules.

23796 For example, in a makefile target rule to build **lib.a**(**file.o**), $% represents **file.o**, as
23797 opposed to $@, which represents **lib.a**.

23798 $? The $? macro shall evaluate to the list of prerequisites that are newer than the current
23799 target. It shall be evaluated for both target and inference rules.

23800 For example, in a makefile target rule to build *prog* from **file1.o**, **file2.o**, and **file3.o**, and
23801 where *prog* is not out of date with respect to **file1.o**, but is out of date with respect to
23802 **file2.o** and **file3.o**, $? represents **file2.o** and **file3.o**.

23803 $< In an inference rule, the $< macro shall evaluate to the filename whose existence
23804 allowed the inference rule to be chosen for the target. In the **.DEFAULT** rule, the $<
23805 macro shall evaluate to the current target name. The meaning of the $< macro shall be |
23806 otherwise unspecified. |

23807 For example, in the **.c.a** inference rule, $< represents the prerequisite **.c** file.

23808 $* The $* macro shall evaluate to the current target name with its suffix deleted. It shall be
23809 evaluated at least for inference rules.

23810 For example, in the **.c.a** inference rule, $*.o represents the out-of-date **.o** file that
23811 corresponds to the prerequisite **.c** file.

23812 Each of the internal macros has an alternative form. When an uppercase 'D' or 'F' is appended |
23813 to any of the macros, the meaning shall be changed to the *directory part* for 'D' and *filename part* |
23814 for 'F'. The directory part is the path prefix of the file without a trailing slash; for the current
23815 directory, the directory part is '.'. When the $? macro contains more than one prerequisite
23816 filename, the $(?D) and $(?F) (or ${?D} and ${?F}) macros expand to a list of directory name parts
23817 and filename parts respectively.

23818 For the target *lib*(*member***.o**) and the **s2.a** rule, the internal macros shall be defined as: |

23819 $< *member***.s2**

23820 $* *member*

23821 $@ *lib*

23822 $? *member***.s2**

23823 $% *member***.o**

23824 **Default Rules**

23825 The default rules for *make* shall achieve results that are the same as if the following were used.
23826 Implementations that do not support the C-Language Development Utilities option may omit
23827 **CC**, **CFLAGS**, **YACC**, **YFLAGS**, **LEX**, **LFLAGS**, **LDFLAGS**, and the **.c**, **.y**, and **.l** inference rules.
23828 Implementations that do not support FORTRAN may omit **FC**, **FFLAGS**, and the **.f** inference
23829 rules. Implementations may provide additional macros and rules.

23830 *SPECIAL TARGETS*

23831 XSI
```
.SCCS_GET: sccs $(SCCSFLAGS) get $(SCCSGETFLAGS) $@
```
23832

23833 XSI
```
.SUFFIXES: .o .c .y .l .a .sh .f .c~ .y~ .l~ .sh~ .f~
```

23834 *MACROS*

```
MAKE=make
AR=ar
ARFLAGS=-rv
YACC=yacc
YFLAGS=
LEX=lex
LFLAGS=
LDFLAGS=
CC=c99
CFLAGS=-O
FC=fort77
```

```
23846         FFLAGS=-O 1
23847 XSI     GET=get
23848         GFLAGS=
23849         SCCSFLAGS=
23850         SCCSGETFLAGS=-s
23851
23852         SINGLE SUFFIX RULES
23853         .c:
23854             $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $<
23855         .f:
23856             $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $<
23857         .sh:
23858             cp $< $@
23859             chmod a+x $@
23860 XSI     .c~:
23861             $(GET) $(GFLAGS) -p $< > $*.c
23862             $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $*.c
23863         .f~:
23864             $(GET) $(GFLAGS) -p $< > $*.f
23865             $(FC) $(FFLAGS) $(LDFLAGS) -o $@ $*.f
23866         .sh~:
23867             $(GET) $(GFLAGS) -p $< > $*.sh
23868             cp $*.sh $@
23869             chmod a+x $@
23870
23871         DOUBLE SUFFIX RULES
23872         .c.o:
23873             $(CC) $(CFLAGS) -c $<
23874         .f.o:
23875             $(FC) $(FFLAGS) -c $<
23876         .y.o:
23877             $(YACC) $(YFLAGS) $<
23878             $(CC) $(CFLAGS) -c y.tab.c
23879             rm -f y.tab.c
23880             mv y.tab.o $@
23881         .l.o:
23882             $(LEX) $(LFLAGS) $<
23883             $(CC) $(CFLAGS) -c lex.yy.c
23884             rm -f lex.yy.c
23885             mv lex.yy.o $@
23886         .y.c:
23887             $(YACC) $(YFLAGS) $<
23888             mv y.tab.c $@
23889         .l.c:
23890             $(LEX) $(LFLAGS) $<
```

```
        mv lex.yy.c $@

XSI .c~.o:
        $(GET) $(GFLAGS) –p $< > $*.c
        $(CC) $(CFLAGS) –c $*.c

    .f~.o:
        $(GET) $(GFLAGS) –p $< > $*.f
        $(FC) $(FFLAGS) –c $*.f

    .y~.o:
        $(GET) $(GFLAGS) –p $< > $*.y
        $(YACC) $(YFLAGS) $*.y
        $(CC) $(CFLAGS) –c y.tab.c
        rm –f y.tab.c
        mv y.tab.o $@

    .l~.o:
        $(GET) $(GFLAGS) –p $< > $*.l
        $(LEX) $(LFLAGS) $*.l
        $(CC) $(CFLAGS) –c lex.yy.c
        rm –f lex.yy.c
        mv lex.yy.o $@

    .y~.c:
        $(GET) $(GFLAGS) –p $< > $*.y
        $(YACC) $(YFLAGS) $*.y
        mv y.tab.c $@

    .l~.c:
        $(GET) $(GFLAGS) –p $< > $*.l
        $(LEX) $(LFLAGS) $*.l
        mv lex.yy.c $@

    .c.a:
        $(CC) –c $(CFLAGS) $<
        $(AR) $(ARFLAGS) $@ $*.o
        rm –f $*.o

    .f.a:
        $(FC) –c $(FFLAGS) $<
        $(AR) $(ARFLAGS) $@ $*.o
        rm –f $*.o
```

EXIT STATUS

When the **–q** option is specified, the *make* utility shall exit with one of the following values:

0 Successful completion.

1 The target was not up-to-date.

>1 An error occurred.

When the **–q** option is not specified, the *make* utility shall exit with one of the following values:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

If there is a source file (such as **./source.c**) and there are two SCCS files corresponding to it (**./s.source.c** and **./SCCS/s.source.c**), on XSI-conformant systems *make* uses the SCCS file in the current directory. However, users are advised to use the underlying SCCS utilities (*admin*, *delta*, *get*, and so on) or the *sccs* utility for all source files in a given directory. If both forms are used for a given source file, future developers are very likely to be confused.

It is incumbent upon portable makefiles to specify the **.POSIX** special target in order to guarantee that they are not affected by local extensions.

The –**k** and –**S** options are both present so that the relationship between the command line, the *MAKEFLAGS* variable, and the makefile can be controlled precisely. If the **k** flag is passed in *MAKEFLAGS* and a command is of the form:

```
$(MAKE) –S foo
```

then the default behavior is restored for the child *make*.

When the –**n** option is specified, it is always added to *MAKEFLAGS*. This allows a recursive *make* –**n** *target* to be used to see all of the action that would be taken to update *target*.

Because of widespread historical practice, interpreting a `'#'` number sign inside a variable as the start of a comment has the unfortunate side effect of making it impossible to place a number sign in a variable, thus forbidding something like:

```
CFLAGS = "–D COMMENT_CHAR='#'"
```

Many historical *make* utilities stop chaining together inference rules when an intermediate target is nonexistent. For example, it might be possible for a *make* to determine that both **.y.c** and **.c.o** could be used to convert a **.y** to a **.o**. Instead, in this case, *make* requires the use of a **.y.o** rule.

The best way to provide portable makefiles is to include all of the rules needed in the makefile itself. The rules provided use only features provided by other parts of this volume of IEEE Std 1003.1-200x. The default rules include rules for optional commands in this volume of IEEE Std 1003.1-200x. Only rules pertaining to commands that are provided are needed in an implementation's default set.

Macros used within other macros are evaluated when the new macro is used rather than when the new macro is defined. Therefore:

```
MACRO = value1
NEW   = $(MACRO)
MACRO = value2

target:
    echo $(NEW)
```

would produce *value2* and not *value1* since **NEW** was not expanded until it was needed in the *echo* command line.

Some historical applications have been known to intermix *target_name* and *macro=name* operands on the command line, expecting that all of the macros are processed before any of the targets are dealt with. Conforming applications do not do this, although some backward compatibility support may be included in some implementations.

The following characters in filenames may give trouble: `'='`, `':'`, `'''`, `'''`, and `'@'`. For inference rules, the description of $< and $? seem similar. However, an example shows the

minor difference. In a makefile containing:

```
foo.o: foo.h
```

if **foo.h** is newer than **foo.o**, yet **foo.c** is older than **foo.o**, the built-in rule to make **foo.o** from **foo.c** is used, with $< equal to **foo.c** and $? equal to **foo.h**. If **foo.c** is also newer than **foo.o**, $< is equal to **foo.c** and $? is equal to **foo.h foo.c**.

EXAMPLES

1. The following command:

   ```
   make
   ```

   makes the first target found in the makefile.

2. The following command:

   ```
   make junk
   ```

   makes the target **junk**.

3. The following makefile says that **pgm** depends on two files, **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**), and a common file **incl.h**:

   ```
   pgm: a.o b.o
       c99 a.o b.o –o pgm
   a.o: incl.h a.c
       c99 –c a.c
   b.o: incl.h b.c
       c99 –c b.c
   ```

4. An example for making optimized **.o** files from **.c** files is:

   ```
   .c.o:
       c99 –c –O $*.c
   ```

   or:

   ```
   .c.o:
       c99 –c –O $<
   ```

5. The most common use of the archive interface follows. Here, it is assumed that the source files are all C-language source:

   ```
   lib: lib(file1.o) lib(file2.o) lib(file3.o)
       @echo lib is now up-to-date
   ```

   The **.c.a** rule is used to make **file1.o**, **file2.o**, and **file3.o** and insert them into **lib**.

   The treatment of escaped <newline>s throughout the makefile is historical practice. For example, the inference rule:

   ```
   .c.o\
   :
   ```

   works, and the macro:

   ```
   f=  bar baz\
       biz
   a:
       echo ==$f==
   ```

24019 echoes "==bar baz biz==".

24020 If $? were:

24021 `/usr/include/stdio.h /usr/include/unistd.h foo.h`

24022 then $(?D) would be:

24023 `/usr/include /usr/include .`

24024 and $(?F) would be:

24025 `stdio.h unistd.h foo.h`

24026 6. The contents of the built-in rules can be viewed by running:

24027 `make –p –f /dev/null 2>/dev/null`

24028 **RATIONALE**

24029 The *make* utility described in this volume of IEEE Std 1003.1-200x is intended to provide the
24030 means for changing portable source code into executables that can be run on a
24031 IEEE Std 1003.1-200x-conforming system. It reflects the most common features present in
24032 System V and BSD *make*s.

24033 Historically, the *make* utility has been an especially fertile ground for vendor and research
24034 organization-specific syntax modifications and extensions. Examples include:

24035 • Syntax supporting parallel execution (such as from various multi-processor vendors, GNU,
24036 and others)

24037 • Additional ‘‘operators’’ separating targets and their prerequisites (System V, BSD, and
24038 others)

24039 • Specifying that command lines containing the strings "${MAKE}" and "$(MAKE)" are
24040 executed when the **–n** option is specified (GNU and System V)

24041 • Modifications of the meaning of internal macros when referencing libraries (BSD and others)

24042 • Using a single instance of the shell for all of the command lines of the target (BSD and others)

24043 • Allowing spaces as well as tabs to delimit command lines (BSD)

24044 • Adding C preprocessor-style ‘‘include’’ and ‘‘ifdef’’ constructs (System V, GNU, BSD, and
24045 others)

24046 • Remote execution of command lines (Sprite and others)

24047 • Specifying additional special targets (BSD, System V, and most others)

24048 Additionally, many vendors and research organizations have rethought the basic concepts of
24049 *make*, creating vastly extended, as well as completely new, syntaxes. Each of these versions of
24050 *make* fulfills the needs of a different community of users; it is unreasonable for this volume of
24051 IEEE Std 1003.1-200x to require behavior that would be incompatible (and probably inferior) to
24052 historical practice for such a community.

24053 In similar circumstances, when the industry has enough sufficiently incompatible formats as to
24054 make them irreconcilable, this volume of IEEE Std 1003.1-200x has followed one or both of two
24055 courses of action. Commands have been renamed (*cksum*, *echo*, and *pax*) and/or command line
24056 options have been provided to select the desired behavior (*grep*, *od*, and *pax*).

24057 Because the syntax specified for the *make* utility is, by and large, a subset of the syntaxes
24058 accepted by almost all versions of *make*, it was decided that it would be counter-productive to
24059 change the name. And since the makefile itself is a basic unit of portability, it would not be

completely effective to reserve a new option letter, such as *make* –**P**, to achieve the portable behavior. Therefore, the special target **.POSIX** was added to the makefile, allowing users to specify ‘‘standard’’ behavior. This special target does not preclude extensions in the *make* utility, nor does it preclude such extensions being used by the makefile specifying the target; it does, however, preclude any extensions from being applied that could alter the behavior of previously valid syntax; such extensions must be controlled via command line options or new special targets. It is incumbent upon portable makefiles to specify the **.POSIX** special target in order to guarantee that they are not affected by local extensions.

The portable version of *make* described in this reference page is not intended to be the state-of-the-art software generation tool and, as such, some newer and more leading-edge features have not been included. An attempt has been made to describe the portable makefile in a manner that does not preclude such extensions as long as they do not disturb the portable behavior described here.

When the –**n** option is specified, it is always added to *MAKEFLAGS*. This allows a recursive *make* –**n** *target* to be used to see all of the action that would be taken to update *target*.

The definition of *MAKEFLAGS* allows both the System V letter string and the BSD command line formats. The two formats are sufficiently different to allow implementations to support both without ambiguity.

Early proposals stated that an ‘‘unquoted’’ number sign was treated as the start of a comment. The *make* utility does not pay any attention to quotes. A number sign starts a comment regardless of its surroundings.

The text about ‘‘other implementation-defined pathnames may also be tried’’ in addition to **./makefile** and **./Makefile** is to allow such extensions as **SCCS/s.Makefile** and other variations. It was made an implementation-defined requirement (as opposed to unspecified behavior) to highlight surprising implementations that might select something unexpected like **/etc/Makefile**. XSI-conformant systems also try **./s.makefile**, **SCCS/s.makefile**, **./s.Makefile**, and **SCCS/s.Makefile**.

Early proposals contained the macro **NPROC** as a means of specifying that *make* should use *n* processes to do the work required. While this feature is a valuable extension for many systems, it is not common usage and could require other non-trivial extensions to makefile syntax. This extension is not required by this volume of IEEE Std 1003.1-200x, but could be provided as a compatible extension. The macro **PARALLEL** is used by some historical systems with essentially the same meaning (but without using a name that is a common system limit value). It is suggested that implementors recognize the existing use of **NPROC** and/or **PARALLEL** as extensions to *make*.

The default rules are based on System V. The default **CC=** value is *c99* instead of *cc* because this volume of IEEE Std 1003.1-200x does not standardize the utility named *cc*. Thus, every conforming application would be required to define **CC=***c99* to expect to run. There is no advantage conferred by the hope that the makefile might hit the ‘‘preferred’’ compiler because this cannot be guaranteed to work. Also, since the portable makescript can only use the *c99* options, no advantage is conferred in terms of what the script can do. It is a quality-of-implementation issue as to whether *c99* is as valuable as *cc*.

The –**d** option to *make* is frequently used to produce debugging information, but is too implementation-defined to add to this volume of IEEE Std 1003.1-200x.

The –**p** option is not passed in *MAKEFLAGS* on most historical implementations and to change this would cause many implementations to break without sufficiently increased portability.

Commands that begin with a plus sign (`'+'`) are executed even if the **–n** option is present. Based on the GNU version of *make*, the behavior of **–n** when the plus-sign prefix is encountered has been extended to apply to **–q** and **–t** as well. However, the System V convention of forcing command execution with **–n** when the command line of a target contains either of the strings `"$(MAKE)"` or `"${MAKE}"` has not been adopted. This functionality appeared in early proposals, but the danger of this approach was pointed out with the following example of a portion of a makefile:

```
subdir:
    cd subdir; rm all_the_files; $(MAKE)
```

The loss of the System V behavior in this case is well-balanced by the safety afforded to other makefiles that were not aware of this situation. In any event, the command line plus-sign prefix can provide the desired functionality.

The double colon in the target rule format is supported in BSD systems to allow more than one target line containing the same target name to have commands associated with it. Since this is not functionality described in the SVID or XPG3 it has been allowed as an extension, but not mandated.

The default rules are provided with text specifying that the built-in rules shall be the same *as if* the listed set were used. The intent is that implementations should be able to use the rules without change, but will be allowed to alter them in ways that do not affect the primary behavior.

The best way to provide portable makefiles is to include all of the rules needed in the makefile itself. The rules provided use only features provided by other portions of this volume of IEEE Std 1003.1-200x. The default rules include rules for optional commands in this volume of IEEE Std 1003.1-200x. Only rules pertaining to commands that are provided are needed in the default set of an implementation.

One point of discussion was whether to drop the default rules list from this volume of IEEE Std 1003.1-200x. They provide convenience, but do not enhance portability of applications. The prime benefit is in portability of users who wish to type *make command* and have the command build from a **command.c** file.

The historical *MAKESHELL* feature was omitted. In some implementations it is used to let a user override the shell to be used to run *make* commands. This was confusing; for a portable *make*, the shell should be chosen by the makefile writer or specified on the *make* command line and not by a user running *make*.

The *make* utilities in most historical implementations process the prerequisites of a target in left-to-right order, and the makefile format requires this. It supports the standard idiom used in many makefiles that produce *yacc* programs; for example:

```
foo: y.tab.o lex.o main.o
    $(CC) $(CFLAGS) –o $@ t.tab.o lex.o main.o
```

In this example, if *make* chose any arbitrary order, the **lex.o** might not be made with the correct **y.tab.h**. Although there may be better ways to express this relationship, it is widely used historically. Implementations that desire to update prerequisites in parallel should require an explicit extension to *make* or the makefile format to accomplish it, as described previously.

The algorithm for determining a new entry for target rules is partially unspecified. Some historical *make*s allow blank, empty, or comment lines within the collection of commands marked by leading <tab>s. A conforming makefile must ensure that each command starts with a <tab>, but implementations are free to ignore blank, empty, and comment lines without triggering the start of a new entry.

The ASYNCHRONOUS EVENTS section includes having SIGTERM and SIGHUP, along with the more traditional SIGINT and SIGQUIT, remove the current target unless directed not to do so. SIGTERM and SIGHUP were added to parallel other utilities that have historically cleaned up their work as a result of these signals. When *make* receives any signal other than SIGQUIT, it is required to resend itself the signal it received so that it exits with a status that reflects the signal. The results from SIGQUIT are partially unspecified because, on systems that create **core** files upon receipt of SIGQUIT, the **core** from *make* would conflict with a core file from the command that was running when the SIGQUIT arrived. The main concern was to prevent damaged files from appearing up-to-date when *make* is rerun.

The **.PRECIOUS** special target was extended to affect all targets globally (by specifying no prerequisites). The **.IGNORE** and **.SILENT** special targets were extended to allow prerequisites; it was judged to be more useful in some cases to be able to turn off errors or echoing for a list of targets than for the entire makefile. These extensions to the *make* in System V were made to match historical practice from the BSD *make*.

Macros are not exported to the environment of commands to be run. This was never the case in any historical *make* and would have serious consequences. The environment is the same as the environment to *make* except that *MAKEFLAGS* and macros defined on the *make* command line are added.

Some implementations do not use *system*( ) for all command lines, as required by the portable makefile format; as a performance enhancement, they select lines without shell metacharacters for direct execution by *execve*( ). There is no requirement that *system*( ) be used specifically, but merely that the same results be achieved. The metacharacters typically used to bypass the direct *execve*( ) execution have been any of:

```
=  |  ^  (  )  ;  &  <  >  *  ?  [  ]  :  $  ‘  ’  "  \  \n
```

The default in some advanced versions of *make* is to group all the command lines for a target and execute them using a single shell invocation; the System V method is to pass each line individually to a separate shell. The single-shell method has the advantages in performance and the lack of a requirement for many continued lines. However, converting to this newer method has caused portability problems with many historical makefiles, so the behavior with the POSIX makefile is specified to be the same as that of System V. It is suggested that the special target **.ONESHELL** be used as an implementation extension to achieve the single-shell grouping for a target or group of targets.

Novice users of *make* have had difficulty with the historical need to start commands with a <tab>. Since it is often difficult to discern differences between <tab>s and <space>s on terminals or printed listings, confusing bugs can arise. In early proposals, an attempt was made to correct this problem by allowing leading <blank>s instead of <tab>s. However, implementors reported many makefiles that failed in subtle ways following this change, and it is difficult to implement a *make* that unambiguously can differentiate between macro and command lines. There is extensive historical practice of allowing leading spaces before macro definitions. Forcing macro lines into column 1 would be a significant backwards-compatibility problem for some makefiles. Therefore, historical practice was restored.

The System V INCLUDE feature was considered, but not included. This would treat a line that began in the first column and contained INCLUDE <*filename*> as an indication to read <*filename*> at that point in the makefile. This is difficult to use in a portable way, and it raises concerns about nesting levels and diagnostics. System V, BSD, GNU, and others have used different methods for including files.

The System V dynamic dependency feature was not included. It would support:

```
cat: $$@.c
```

that would expand to;

```
cat: cat.c
```

This feature exists only in the new version of System V *make* and, while useful, is not in wide usage. This means that macros are expanded twice for prerequisites: once at makefile parse time and once at target update time.

Consideration was given to adding metarules to the POSIX *make*. This would make **%.o: %.c** the same as **.c.o:**. This is quite useful and available from some vendors, but it would cause too many changes to this *make* to support. It would have introduced rule chaining and new substitution rules. However, the rules for target names have been set to reserve the '%' and '"' characters. These are traditionally used to implement metarules and quoting of target names, respectively. Implementors are strongly encouraged to use these characters only for these purposes.

A request was made to extend the suffix delimiter character from a period to any character. The metarules feature in newer *make*s solves this problem in a more general way. This volume of IEEE Std 1003.1-200x is staying with the more conservative historical definition.

The standard output format for the **–p** option is not described because it is primarily a debugging option and because the format is not generally useful to programs. In historical implementations the output is not suitable for use in generating makefiles. The **–p** format has been variable across historical implementations. Therefore, the definition of **–p** was only to provide a consistently named option for obtaining *make* script debugging information.

Some historical implementations have not cleared the suffix list with **–r**.

Implementations should be aware that some historical applications have intermixed *target_name* and *macro=value* operands on the command line, expecting that all of the macros are processed before any of the targets are dealt with. Conforming applications do not do this, but some backwards-compatibility support may be warranted.

Empty inference rules are specified with a semicolon command rather than omitting all commands, as described in an early proposal. The latter case has no traditional meaning and is reserved for implementation extensions, such as in GNU *make*.

FUTURE DIRECTIONS

None.

SEE ALSO

ar, *c99*, *get*, *lex*, *sh*, *yacc*, the System Interfaces volume of IEEE Std 1003.1-200x, *system*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

FUTURE DIRECTIONS section added.

Issue 6

This utility is now marked as part of the Software Development Utilities option.

The Open Group Corrigendum U029/1 is applied, correcting a typographical error in the SPECIAL TARGETS section.

In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from ''otherwise, the home directory of a user of that name is examined'' to ''otherwise, the value of *PROJECTDIR* is treated as a user name and that user's initial working directory is examined''.

It is specified whether the command line is related to the makefile or to the *make* command, and the macro processing rules are updated to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

PASC Interpretation 1003.2 #193 is applied.

NAME

man — display system documentation

SYNOPSIS

```
man [–k] name...
```

DESCRIPTION

The *man* utility shall write information about each of the *name* operands. If *name* is the name of a standard utility, *man* at a minimum shall 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.

OPTIONS

The *man* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following option shall be supported:

–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 single entry per line; this organization is not required and the example using *grep* **–Ei** is merely illustrative of the type of search intended. The purpose entry to be included in the database shall consist of a terse description of the purpose of the utility.

OPERANDS

The following operand shall be supported:

name A keyword or the name of a standard utility. When **–k** is not specified and *name* does not represent one of the standard utilities, the results are unspecified.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *man*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and in the summary database). The value of *LC_CTYPE* need not affect the format of the information written about the name operands.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

PAGER Determine an output filtering command for writing the output to a terminal. Any string acceptable as a *command_string* operand to the *sh* **–c** command shall be valid. When standard output is a terminal device, the reference page output shall be piped through the command. If the *PAGER* variable is null or not set, the command shall be either *more* or another paginator utility documented in the system documentation.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The *man* utility shall write 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.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

It is recognized that the *man* utility is only of minimal usefulness as specified. The opinion of the standard developers was strongly divided as to how much or how little information *man* should be required to provide. They considered, however, that the provision of some portable way of accessing documentation would aid user portability. The arguments against a fuller

specification were:

- Large quantities of documentation should not be required on a system that does not have excess disk space.
- The current manual system does not present information in a manner that greatly aids user portability.
- 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 this volume of IEEE Std 1003.1-200x.

The description was changed to be more specific about what has to be displayed for a utility. The standard developers considered it 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 section title (less the numeric prefix) from this volume of IEEE Std 1003.1-200x for each utility. These titles are similar to those used in historical systems for this purpose.

See *mailx* for rationale concerning the default paginator.

The caveat in the *LC_CTYPE* description was added because it is not a requirement that an implementation provide reference pages for all of its supported locales on each system; changing *LC_CTYPE* does not necessarily translate the reference page into another language. This is equivalent to the current state of *LC_MESSAGES* in IEEE Std 1003.1-200x—locale-specific messages are not yet a requirement.

The historical *MANPATH* variable is not included in POSIX because no attempt is made to specify naming conventions for reference page files, nor even to mandate that they are files at all. On some implementations they could be a true database, a hypertext file, or even fixed strings within the *man* executable. The standard developers considered the portability of reference pages to be outside their scope of work. 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 reference pages from the various categories (utilities, functions, file formats, and so on) when the system administrator reveals the location and conventions for reference pages on the system.

The keyword search can rely on at least the text of the section titles from these utility descriptions, and the implementation may add more keywords. The term ‘‘section titles’’ refers to the strings such as:

```
man — Display system documentation
ps — Report process status
```

FUTURE DIRECTIONS

None.

SEE ALSO

more

CHANGE HISTORY

First released in Issue 4.

Issue 5

FUTURE DIRECTIONS section added.

NAME

mesg — permit or deny messages

SYNOPSIS

UP

```
mesg [y|n]
```

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 affected shall be determined by searching for the first terminal in the sequence of devices associated with standard input, standard output, and standard error, respectively. With no arguments, *mesg* shall report the current state without changing it. Processes with appropriate privileges may be able to send messages to the terminal independent of the current state.

OPTIONS

None.

OPERANDS

The following operands shall be supported in the POSIX locale:

y — Grant permission to other users to send messages to the terminal device.

n — Deny permission to other users to send messages to the terminal device.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *mesg*:

LANG — Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL — If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE — Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES — Determine the locale that should be used to affect the format and contents of diagnostic messages written (by *mesg*) to standard error.

XSI *NLSPATH* — Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

If no operand is specified, *mesg* shall display the current terminal state in an unspecified format.

24419 **STDERR**

24420 The standard error shall be used only for diagnostic messages.

24421 **OUTPUT FILES**

24422 None.

24423 **EXTENDED DESCRIPTION**

24424 None.

24425 **EXIT STATUS**

24426 The following exit values shall be returned:

24427 0 Receiving messages is allowed.

24428 1 Receiving messages is not allowed.

24429 >1 An error occurred.

24430 **CONSEQUENCES OF ERRORS**

24431 Default.

24432 **APPLICATION USAGE**

24433 The mechanism by which the message status of the terminal is changed is unspecified.
24434 Therefore, unspecified actions may cause the status of the terminal to change after *mesg* has
24435 successfully completed. These actions may include, but are not limited to: another invocation of
24436 the *mesg* utility, login procedures; invocation of the *stty* utility, invocation of the *chmod* utility or
24437 *chmod*( ) function, and so on.

24438 **EXAMPLES**

24439 None.

24440 **RATIONALE**

24441 The terminal changed by *mesg* is that associated with the standard input, output, or error, rather
24442 than the controlling terminal for the session. This is because users logged in more than once
24443 should be able to change any of their login terminals without having to stop the job running in
24444 those sessions. This is not a security problem involving the terminals of other users because
24445 appropriate privileges would be required to affect the terminal of another user.

24446 The method of checking each of the first three file descriptors in sequence until a terminal is
24447 found was adopted from System V.

24448 The file **/dev/tty** is not specified for the terminal device because it was thought to be too
24449 restrictive. Typical environment changes for the **n** operand are that write permissions are
24450 removed for *others* and *group* from the appropriate device. It was decided to leave the actual
24451 description of what is done as unspecified because of potential differences between
24452 implementations.

24453 The format for standard output is unspecified because of differences between historical
24454 implementations. This output is generally not useful to shell scripts (they can use the exit
24455 status), so exact parsing of the output is unnecessary.

24456 **FUTURE DIRECTIONS**

24457 None.

24458 **SEE ALSO**

24459 *talk*, *write*

24460 **CHANGE HISTORY**

24461 First released in Issue 2.

24462 **Issue 6**

24463 This utility is now marked as part of the User Portability Utilities option.

NAME

mkdir — make directories

SYNOPSIS

```
mkdir [–p][–m mode] dir...
```

DESCRIPTION

The *mkdir* utility shall create the directories specified by the operands, in the order specified.

For each *dir* operand, the *mkdir* utility shall perform actions equivalent to the *mkdir*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x, called with the following arguments:

1. The *dir* operand is used as the *path* argument.
2. The value of the bitwise-inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO is used as the *mode* argument. (If the **–m** option is specified, the *mode* option-argument overrides this default.)

OPTIONS

The *mkdir* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–m *mode* Set the file permission bits of the newly-created directory to the specified *mode* value. The *mode* option-argument shall be the same as the *mode* operand defined for the *chmod* utility. In the *symbolic_mode* strings, the *op* characters `'+'` and `'–'` shall be interpreted relative to an assumed initial mode of *a=rwx*; `'+'` shall add permissions to the default mode, `'–'` shall delete permissions from the default mode.

–p Create any missing intermediate pathname components.

For each *dir* operand that does not name an existing directory, effects equivalent to those caused by the following command shall occur:

```
mkdir –p –m $(umask –S),u+wx $(dirname dir) &&
mkdir [–m mode] dir
```

where the **–m** *mode* option represents that option supplied to the original invocation of *mkdir*, if any.

Each *dir* operand that names an existing directory shall be ignored without error.

OPERANDS

The following operand shall be supported:

dir A pathname of a directory to be created.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *mkdir*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables

used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
Not used.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 All the specified directories were created successfully or the **–p** option was specified and all the specified directories now exist.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
The default file mode for directories is *a=rwx* (777 on most systems) with selected permissions removed in accordance with the file mode creation mask. For intermediate pathname components created by *mkdir*, the mode is the default modified by *u+wx* so that the subdirectories can always be created regardless of the file mode creation mask; if different ultimate permissions are desired for the intermediate directories, they can be changed afterwards with *chmod*.

Note that some of the requested directories may have been created even if an error occurs.

EXAMPLES
None.

RATIONALE
The System V **–m** option was included to control the file mode.

The System V **–p** option was included to create any needed intermediate directories and to complement the functionality provided by *rmdir* for removing directories in the path prefix as they become empty. Because no error is produced if any path component already exists, the **–p** option is also useful to ensure that a particular directory exists.

The functionality of *mkdir* is described substantially through a reference to the *mkdir*( ) function in the System Interfaces volume of IEEE Std 1003.1-200x. For example, by default, the mode of the directory is affected by the file mode creation mask in accordance with the specified behavior of the *mkdir*( ) function. In this way, there is less duplication of effort required for describing details of the directory creation.

FUTURE DIRECTIONS

None.

SEE ALSO

rm, *rmdir*, *umask*, the System Interfaces volume of IEEE Std 1003.1-200x, *mkdir*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

FUTURE DIRECTIONS section added.

NAME

mkfifo — make FIFO special files

SYNOPSIS

```
mkfifo [–m mode] file...
```

DESCRIPTION

The *mkfifo* utility shall create the FIFO special files specified by the operands, in the order specified.

For each *file* operand, the *mkfifo* utility shall perform actions equivalent to the *mkfifo*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x, called with the following arguments:

1. The *file* operand is used as the *path* argument.
2. The value of the bitwise-inclusive OR of S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH is used as the *mode* argument. (If the –**m** option is specified, the value of the *mkfifo*( ) *mode* argument is unspecified, but the FIFO shall at no time have permissions less restrictive than the –**m** *mode* option-argument.)

OPTIONS

The *mkfifo* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following option shall be supported:

–**m** *mode* Set the file permission bits of the newly-created FIFO to the specified *mode* value. The *mode* option-argument shall be the same as the *mode* operand defined for the *chmod* utility. In the *symbolic_mode* strings, the *op* characters `'+'` and `'–'` shall be interpreted relative to an assumed initial mode of *a=rw*.

OPERANDS

The following operand shall be supported:

file A pathname of the FIFO special file to be created.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *mkfifo*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

24607 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

24608 **ASYNCHRONOUS EVENTS**
24609 Default.

24610 **STDOUT**
24611 Not used.

24612 **STDERR**
24613 The standard error shall be used only for diagnostic messages.

24614 **OUTPUT FILES**
24615 None.

24616 **EXTENDED DESCRIPTION**
24617 None.

24618 **EXIT STATUS**
24619 The following exit values shall be returned:

24620 0 All the specified FIFO special files were created successfully.

24621 >0 An error occurred.

24622 **CONSEQUENCES OF ERRORS**
24623 Default.

24624 **APPLICATION USAGE**
24625 None.

24626 **EXAMPLES**
24627 None.

24628 **RATIONALE**
24629 This utility was added to permit shell applications to create FIFO special files.

24630 The **–m** option was added to control the file mode, for consistency with the similar functionality
24631 provided the *mkdir* utility.

24632 Early proposals included a **–p** option similar to the *mkdir* **–p** option that created intermediate
24633 directories leading up to the FIFO specified by the final component. This was removed because
24634 it is not commonly needed and is not common practice with similar utilities.

24635 The functionality of *mkfifo* is described substantially through a reference to the *mkfifo*( ) function
24636 in the System Interfaces volume of IEEE Std 1003.1-200x. For example, by default, the mode of
24637 the FIFO file is affected by the file mode creation mask in accordance with the specified behavior
24638 of the *mkfifo*( ) function. In this way, there is less duplication of effort required for describing
24639 details of the file creation.

24640 **FUTURE DIRECTIONS**
24641 None.

24642 **SEE ALSO**
24643 *umask*, the System Interfaces volume of IEEE Std 1003.1-200x, *mkfifo*( )

24644 **CHANGE HISTORY**
24645 First released in Issue 3.

NAME

more — display files on a page-by-page basis

SYNOPSIS

UP

```
more [–ceisu][–n number][–p command][–t tagstring][file ...]
```

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, without modification, except as specified for the **–s** option. If standard output is a terminal device, the files shall be written a number of lines (one screenful) at a time under the control of user commands. See the EXTENDED DESCRIPTION section.

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
- Shall, 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
- Shall not be required for implementations supporting only fully capable terminals
- Shall not affect commands already requiring <newline>s
- Shall not affect users on the capable terminals from using *more* as described in this volume of IEEE Std 1003.1-200x

OPTIONS

The *more* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–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 silently ignored on devices with insufficient terminal capabilities.

–e By default, *more* shall exit immediately after writing the last line of the last file in the argument list. If the **–e** option is specified:

1. If there is only a single file in the argument list and that file was completely displayed on a single screen, *more* shall exit immediately after writing the last line of that file.
2. Otherwise, *more* shall exit only after reaching end-of-file on the last file in the argument list twice without an intervening operation. See the EXTENDED DESCRIPTION section.

–i Perform pattern matching in searches without regard to case; see the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.2, Regular Expression General Requirements .

24689 **–n** *number* Specify the number of lines per screenful. The *number* argument is a positive
24690 decimal integer. The **–n** option shall override any values obtained from any other
24691 source.

24692 **–p** *command* Each time a screen from a new file is displayed or redisplayed (including as a
24693 result of *more* commands; for example, **:p**), execute the *more* command(s) in the
24694 command arguments in the order specified, as if entered by the user after the first
24695 screen has been displayed. No intermediate results shall be displayed (that is, if the
24696 command is a movement to a screen different from the normal first screen, only |
24697 the screen resulting from the command shall be displayed.) If any of the |
24698 commands fail for any reason, an informational message to this effect shall be
24699 written, and no further commands specified using the **–p** option shall be executed
24700 for this file.

24701 **–s** Behave as if consecutive empty lines were a single empty line.

24702 **–t** *tagstring* Write the screenful of the file containing the tag named by the *tagstring* argument.
24703 See the *ctags* utility. The tags feature represented by **–t** *tagstring* and the **:t**
24704 command is optional. It shall be provided on any system that also provides a
24705 conforming implementation of *ctags*; otherwise, the use of **–t** produces undefined
24706 results.

24707 The filename resulting from the **–t** option shall be logically added as a prefix to the
24708 list of command line files, as if specified by the user. If the tag named by the
24709 *tagstring* argument is not found, it shall be an error, and *more* shall take no further
24710 action.

24711 If the tag specifies a line number, the first line of the display shall contain the
24712 beginning of that line. If the tag specifies a pattern, the first line of the display shall
24713 contain the beginning of the matching text from the first line of the file that
24714 contains that pattern. If the line does not exist in the file or matching text is not
24715 found, an informational message to this effect shall be displayed, and *more* shall
24716 display the default screen as if **–t** had not been specified.

24717 If both the **–t** *tagstring* and **–p** *command* options are given, the **–t** *tagstring* shall be
24718 processed first; that is, the file and starting line for the display shall be as specified
24719 by **–t**, and then the **–p** *more* command shall be executed. If the line (matching text)
24720 specified by the **–t** command does not exist (is not found), no **–p** *more* command
24721 shall be executed for this file at any time.

24722 **–u** Treat a <backspace> as a printable control character, displayed as an
24723 implementation-defined character sequence (see the EXTENDED DESCRIPTION
24724 section), suppressing backspacing and the special handling that produces
24725 underlined or standout mode text on some terminal types. Also, do not ignore a
24726 <carriage-return> at the end of a line.

24727 **OPERANDS**

24728 The following operand shall be supported:

24729 *file* A pathname of an input file. If no *file* operands are specified, the standard input
24730 shall be used. If a *file* is `'–'`, the standard input shall be read at that point in the
24731 sequence.

24732 **STDIN**

24733 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is `'–'`.

INPUT FILES

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

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *more*:

COLUMNS Override the system-selected horizontal display line size. See the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables for valid values and results when it is unset or null.

EDITOR Used by the **v** command to select an editor. See the EXTENDED DESCRIPTION section.

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes within regular expressions.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

LINES Override the system-selected vertical screen size, used as the number of lines in a screenful. See the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables for valid values and results when it is unset or null. The **–n** option shall take precedence over the *LINES* variable for determining the number of lines in a screenful.

MORE Determine a string containing options described in the OPTIONS section preceded with hyphens and <blank>-separated as on the command line. Any command line options shall be processed after those in the *MORE* variable, as if the command line were:

```
more $MORE options operands
```

The *MORE* variable shall take precedence over the *TERM* and *LINES* variables for determining the number of lines in a screenful.

TERM Determine the name of the terminal type. If this variable is unset or null, an unspecified default terminal type is used.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall be used to write the contents of the input files.

STDERR

The standard error shall be used for diagnostic messages and user commands (see the INPUT FILES section), and, if standard output is a terminal device, to write a prompting string. The prompting string shall appear on the screen line below the last line of the file displayed in the current screenful. The prompt shall contain the name of the file currently being examined and shall contain an end-of-file indication and the name of the next file, if any, when prompting at the end-of-file. If an error or informational message is displayed, it is unspecified whether it is contained in the prompt. If it is not contained in the prompt, it shall be displayed and then the user shall be prompted for a continuation character, at which point another message or the user prompt may be displayed. The prompt is otherwise unspecified. It is unspecified whether informational messages are written for other user commands.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

The following subsection describes the behavior of *more* when the standard output is a terminal device. If the standard output is not a terminal device, no options other than **–s** shall have any effect, and all input files shall be copied to standard output otherwise unmodified, at which time *more* shall exit without further action.

The number of lines available per screen shall be determined by the **–n** option, if present, or by examining values in the environment (see the ENVIRONMENT VARIABLES section). If neither method yields a number, an unspecified number of lines shall be used.

The maximum number of lines written shall be one less than this number, because the screen line after the last line written shall be used to write a user prompt and user input. If the number of lines in the screen is less than two, the results are undefined. It is unspecified whether user input is permitted to be longer than the remainder of the single line where the prompt has been written.

The number of columns available per line shall be determined by examining values in the environment (see the ENVIRONMENT VARIABLES section), with a default value as described in Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

Lines that are longer than the display shall be folded; the length at which folding occurs is unspecified, but should be appropriate for the output device. Folding may occur between glyphs of single characters that take up multiple display columns.

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 sequence of *n* <backspace>s (where *n* is the same as the number of column positions that the character occupies), then by *n* underscore characters ('_'), shall cause that character to be written as underlined text, if the terminal type supports that. The *n* underscore characters, followed first by *n* <backspace>s, then any character with *n* column positions, shall also cause that character to be written as underlined text, if the terminal type supports that.

- A sequence of *n* <backspace>s (where *n* is the same as the number of column positions that the previous character occupies) that appears between two identical printable characters shall cause the first of those two characters to be written as emboldened text (that is, visually brighter, standout mode, or inverse-video mode), if the terminal type supports that, and the second to be discarded. Immediately subsequent occurrences of <backspace>/character pairs for that same character also shall be discarded. (For example, the sequence `"a\ba\ba\ba"` is interpreted as a single emboldened `'a'`.)
- The *more* utility shall logically discard all other <backspace>s from the line as well as the character which precedes them, if any.
- A <carriage-return> at the end of a line shall be ignored, rather than being written as a non-printable character, as described in the next paragraph.

It is implementation-defined how other non-printable characters are written. Implementations should use the same format that they use for the *ex* **print** command; see the OPTIONS section within the *ed* utility. It is unspecified whether a multi-column character shall be separated if it crosses a display line boundary; it shall not be discarded. The behavior is unspecified if the number of columns on the display is less than the number of columns any single character in the line being displayed would occupy.

When each new file is displayed (or redisplayed), *more* shall write the first screen of the file. Once the initial screen has been written, *more* shall prompt for a user command. If the execution of the user command results in a screen that has lines in common with the current screen, and the device has sufficient terminal capabilities, *more* shall scroll the screen; otherwise, it is unspecified whether the screen is scrolled or redrawn.

For all files but the last (including standard input if no file was specified, and for the last file as well, if the **–e** option was not specified), when *more* has written the last line in the file, *more* shall prompt for a user command. This prompt shall contain the name of the next file as well as an indication that *more* has reached end-of-file. If the user command is **f**, <control>-F, <space>, **j**, <newline>, **d**, <control>-D, or **s**, *more* shall display the next file. Otherwise, if displaying the last file, *more* shall exit. Otherwise, *more* shall execute the user command specified.

Several of the commands described in this section display a previous screen from the input stream. In the case that text is being taken from a non-rewindable stream, such as a pipe, it is implementation-defined how much backwards motion is supported. If a command cannot be executed because of a limitation on backwards motion, an error message to this effect shall be displayed, the current screen shall not change, and the user shall be prompted for another command.

If a command cannot be performed because there are insufficient lines to display, *more* shall alert the terminal. If a command cannot be performed because there are insufficient lines to display or a **/** command fails: if the input is the standard input, the last screen in the file may be displayed; otherwise, the current file and screen shall not change, and the user shall be prompted for another command.

The interactive commands in the following sections 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 the Base Definitions volume of IEEE Std 1003.1-200x, Section 9.3, Basic Regular Expressions. The term ‘‘examine’’ is historical usage meaning ‘‘open the file for viewing’’; for example, *more* **foo** would be expressed as examining file **foo**.

In the following descriptions, unless otherwise specified, *line* is a line in the *more* display, not a line from the file being examined.

In the following descriptions, the *current position* refers to two things:

1. The position of the current line on the screen
2. 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.

Help

Synopsis: `h`

Write a summary of these commands and other implementation-defined commands. The behavior shall be as if the *more* utility were executed with the **–e** option on a file that contained the summary information. The user shall be prompted as described earlier in this section when end-of-file is reached. If the user command is one of those specified to continue to the next file, *more* shall return to the file and screen state from which the **h** command was executed.

Scroll Forward One Screenful

Synopsis: `[`*count*`]f`
`[`*count*`]<control>-F`

Scroll forward *count* lines, with a default of one screenful. If *count* is more than the screen size, only the final screenful shall be written.

Scroll Backward One Screenful

Synopsis: `[`*count*`]b`
`[`*count*`]<control>-B`

Scroll backward *count* lines, with a default of one screenful (see the **–n** option). If *count* is more than the screen size, only the final screenful shall be written.

Scroll Forward One Line

Synopsis: `[`*count*`]<space>`
`[`*count*`]j`
`[`*count*`]<newline>`

Scroll forward *count* lines. The default *count* for the <space> shall be one screenful; for **j** and <newline>, one line. The entire *count* lines shall be written, even if *count* is more than the screen size.

Scroll Backward One Line

Synopsis: `[`*count*`]k`

Scroll backward *count* lines. The entire *count* lines shall be written, even if *count* is more than the screen size.

24906 **Scroll Forward One Half Screenful**

24907 *Synopsis*: `[`*count*`]d`
24908 `[`*count*`]<control>-D`

24909 Scroll forward *count* lines, with a default of one half of the screen size. If *count* is specified, it
24910 shall become the new default for subsequent **d**, <control>-D, and **u** commands.

24911 **Skip Forward One Line**

24912 *Synopsis*: `[`*count*`]s`

24913 Display the screenful beginning with the line *count* lines after the last line on the current screen.
24914 If *count* would cause the current position to be such that less than one screenful would be
24915 written, the last screenful in the file shall be written.

24916 **Scroll Backward One Half Screenful**

24917 *Synopsis*: `[`*count*`]u`
24918 `[`*count*`]<control>-U`

24919 Scroll backward *count* lines, with a default of one half of the screen size. If *count* is specified, it
24920 shall become the new default for subsequent **d**, <control>–D, **u**, and <control>–U commands.
24921 The entire *count* lines shall be written, even if *count* is more than the screen size.

24922 **Go to Beginning of File**

24923 *Synopsis*: `[`*count*`]g`

24924 Display the screenful beginning with line *count*.

24925 **Go to End-of-File**

24926 *Synopsis*: `[`*count*`]G`

24927 If *count* is specified, display the screenful beginning with the line *count*. Otherwise, display the
24928 last screenful of the file.

24929 **Refresh the Screen**

24930 *Synopsis*: `r`
24931 `<control>-L`

24932 Refresh the screen.

24933 **Discard and Refresh**

24934 *Synopsis*: `R`

24935 Refresh the screen, discarding any buffered input. If the current file is non-seekable, buffered
24936 input shall not be discarded and the **R** command shall be equivalent to the **r** command. |

Mark Position

Synopsis: `mletter`

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 new file is examined, all marks may be lost.

Return to Mark

Synopsis: `'letter`

Return to the position that was previously marked with the letter named by *letter*, making that line the current position.

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.

Search Forward for Pattern

Synopsis: `[`*count*`]/[!]`*pattern*<newline>

Display the screenful beginning with the *count*th line containing the pattern. The search shall start after the first line currently displayed. The null regular expression (`'/'` followed by a <newline>) shall repeat the search using the previous regular expression, with a default *count*. If the character `'!'` is included, the matching lines shall be those that do not contain the *pattern*. If no match is found for the *pattern*, a message to that effect shall be displayed.

Search Backward for Pattern

Synopsis: `[`*count*`]?[!]`*pattern*<newline>

Display the screenful beginning with the *count*th previous line containing the pattern. The search shall start on the last line before the first line currently displayed. The null regular expression (`'?'` followed by a <newline>) shall repeat the search using the previous regular expression, with a default *count*. If the character `'!'` is included, matching lines shall be those that do not contain the *pattern*.

If no match is found for the *pattern*, a message to that effect shall be displayed.

Repeat Search

Synopsis: `[`*count*`]n`

Repeat the previous search for *count*th line containing the last *pattern* (or not containing the last *pattern*, if the previous search was `"/!"` or `"?!"`).

24970 **Repeat Search in Reverse**

24971 *Synopsis*: `[`*count*`]N`

24972 Repeat the search in the opposite direction of the previous search for the *count*th line containing
24973 the last *pattern* (or not containing the last *pattern*, if the previous search was "/!" or "?!").

24974 **Examine New File**

24975 *Synopsis*: `:e [`*filename*`]`<newline>

24976 Examine a new file. If the *filename* argument is not specified, the current file (see the **:n** and **:p**
24977 commands below) shall be re-examined. The *filename* shall be subjected to the process of shell
24978 word expansions (see Section 2.6 (on page 2238)); if more than a single pathname results, the
24979 effects are unspecified. If *filename* is a number sign ('#'), the previously examined file shall be
24980 re-examined. If *filename* is not accessible for any reason (including that it is a non-seekable file),
24981 an error message to this effect shall be displayed and the current file and screen shall not change.

24982 **Examine Next File**

24983 *Synopsis*: `[`*count*`]:n`

24984 Examine the next file. If a number *count* is specified, the *count*th next file shall be examined. If
24985 *filename* refers to a non-seekable file, the results are unspecified.

24986 **Examine Previous File**

24987 *Synopsis*: `[`*count*`]:p`

24988 Examine the previous file. If a number *count* is specified, the *count*th previous file shall be
24989 examined. If *filename* refers to a non-seekable file, the results are unspecified.

24990 **Go to Tag**

24991 *Synopsis*: `:t` *tagstring*<newline>

24992 If the file containing the tag named by the *tagstring* argument is not the current file, examine the
24993 file, as if the **:e** command was executed with that file as the argument. Otherwise, or in addition,
24994 display the screenful beginning with the tag, as described for the **–t** option (see the OPTIONS
24995 section). If the *ctags* utility is not supported by the system, the use of **:t** produces undefined
24996 results.

24997 **Invoke Editor**

24998 *Synopsis*: `v`

24999 Invoke an editor to edit the current file being examined. If standard input is being examined, the
25000 results are unspecified. The name of the editor shall be taken from the environment variable
25001 *EDITOR*, or shall default to *vi*. If the last pathname component in *EDITOR* is either *vi* or *ex*, the
25002 editor shall be invoked with a **–c** *linenumber* command line argument, where *linenumber* is the
25003 line number of the file line containing the display line currently displayed as the first line of the
25004 screen. It is implementation-defined whether line-setting options are passed to editors other
25005 than *vi* and *ex*.

25006 When the editor exits, *more* shall resume with the same file and screen as when the editor was
25007 invoked.

Display Position

Synopsis: =
<control>-G

Write a message for which the information references the first byte of the line after the last line of the file on the screen. This message shall include the name of the file currently being examined, its number relative to the total number of files there are to examine, the line number in the file, the byte number and the total bytes in the file, 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, the line number, the byte number, the total bytes, and the percentage need not be written.

Quit

Synopsis: q
:q
ZZ

Exit *more*.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

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.

APPLICATION USAGE

When the standard output is not a terminal, only the –**s** filter-modification option is effective. This is based on historical practice. For example, a typical implementation of *man* pipes its output through *more* –**s** to squeeze excess white space for terminal users. When *man* is piped to *lp*, however, it is undesirable for this squeezing to happen.

EXAMPLES

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

RATIONALE

The *more* utility, available in BSD and BSD-derived systems, was chosen as the prototype for the POSIX file display program since it is more widely available than either the public-domain program *less* or than *pg*, a pager provided in System V. The 4.4 BSD *more* is the model for the

features selected; it is almost fully upward-compatible from the 4.3 BSD 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 volume of IEEE Std 1003.1-200x 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.

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 this volume of IEEE Std 1003.1-200x because there are so few systems known to support such terminals. Nevertheless, it was considered 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* **!** command. The standard developers did not consider that this was necessary in a paginator, particularly given the wide acceptance of multiple window terminals and job control features. (They chose to retain such features in the editors and *mailx* because the shell interaction also gives an opportunity to modify the editing buffer, which is not applicable to *more*).

The **–p** (position) option replaces the **+** command because of the Utility Syntax Guidelines. In early proposals, it took a *pattern* argument, but historical *less* provided the *more* general facility of a command. It would have been desirable to use the same **–c** as *ex* and *vi*, but the letter was already in use.

The text stating ‘‘from a non-rewindable stream ... implementations may limit the amount of backwards motion supported’’ would allow an implementation that permitted no backwards motion beyond text already on the screen. It was not possible to require a minimum amount of backwards 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.

Historically, non-printable characters were displayed using the ARPA standard mappings, which are as follows:

1. Printable characters are left alone.
2. Control characters less than \177 are represented as followed by the character offset from the `'@'` character in the ASCII map; for example, \007 is represented as `'G'`.
3. \177 is represented as followed by `'?'`.

The display of characters having their eighth bit set was less standard. Existing implementations use hex (0x00), octal (\000), and a meta-bit display. (The latter displayed characters with their eighth bit set as the two characters `"M–,"` followed by the seven bit display as described previously.) The latter probably has the best claim to historical practice because it was used with the **–v** option of 4 BSD and 4 BSD-derived versions of the *cat* utility since 1980.

No specific display format is required by IEEE Std 1003.1-200x. Implementations are encouraged to conform to historic practice in the absence of any strong reason to diverge.

FUTURE DIRECTIONS

None.

SEE ALSO

ctags, *ed*, *ex*, *vi*

CHANGE HISTORY

First released in Issue 4.

Issue 5

FUTURE DIRECTIONS section added.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The obsolescent SYNOPSIS is removed.

The utility has been extensively reworked for alignment with the IEEE P1003.2b draft standard:

- Changes have been made as result of IEEE PASC Interpretations 1003.2 #37 and #109.
- The *more* utility should be able to handle underlined and emboldened displays of characters that are wider than a single column position.

25115 **NAME**

25116 mv — move files

25117 **SYNOPSIS**

25118 `mv [–fi] source_file target_file`

25119 `mv [–fi] source_file... target_file`

25120 **DESCRIPTION**

25121 In the first synopsis form, the *mv* utility shall move the file named by the *source_file* operand to
25122 the *destination* specified by the *target_file*. This first synopsis form is assumed when the final
25123 operand does not name an existing directory and is not a symbolic link referring to an existing
25124 directory.

25125 In the second synopsis form, *mv* shall move each file named by a *source_file* operand to a
25126 *destination* file in the existing directory named by the *target_dir* operand, or referenced if
25127 *target_dir* is a symbolic link referring to an existing directory. The *destination* path for each
25128 *source_file* shall be the concatenation of the target directory, a single slash character, and the last
25129 pathname component of the *source_file*. This second form is assumed when the final operand
25130 names an existing directory.

25131 If any operand specifies an existing file of a type not specified by the System Interfaces volume
25132 of IEEE Std 1003.1-200x, the behavior is implementation-defined.

25133 For each *source_file* the following steps shall be taken:

25134 1. If the destination path exists, the **–f** option is not specified, and either of the following
25135 conditions is true:

25136 a. The permissions of the destination path do not permit writing and the standard input
25137 is a terminal.

25138 b. The **–i** option is specified.

25139 the *mv* utility shall write a prompt to standard error and read a line from standard input. If
25140 the response is not affirmative, *mv* shall do nothing more with the current *source_file* and
25141 go on to any remaining *source_file*s.

25142 2. The *mv* utility shall perform actions equivalent to the *rename*( ) function defined in the
25143 System Interfaces volume of IEEE Std 1003.1-200x, called with the following arguments:

25144 a. The *source_file* operand is used as the *old* argument.

25145 b. The destination path is used as the *new* argument.

25146 If this succeeds, *mv* shall do nothing more with the current *source_file* and go on to any
25147 remaining *source_file*s. If this fails for any reasons other than those described for the *errno*
25148 [EXDEV] in the System Interfaces volume of IEEE Std 1003.1-200x, *mv* shall write a
25149 diagnostic message to standard error, do nothing more with the current *source_file*, and go
25150 on to any remaining *source_file*s.

25151 3. If the destination path exists, and it is a file of type directory and *source_file* is not a file of
25152 type directory, or it is a file not of type directory and *source_file* is a file of type directory,
25153 *mv* shall write a diagnostic message to standard error, do nothing more with the current
25154 *source_file*, and go on to any remaining *source_file*s.

25155 4. If the destination path exists, *mv* shall attempt to remove it. If this fails for any reason, *mv*
25156 shall write a diagnostic message to standard error, do nothing more with the current
25157 *source_file*, and go on to any remaining *source_file*s.

5. The file hierarchy rooted in *source_file* shall be duplicated as a file hierarchy rooted in the *destination* path. If *source_file* or any of the files below it in the hierarchy are symbolic links, the links themselves shall be duplicated, including their contents, rather than any files to which they refer. The following characteristics of each file in the file hierarchy shall be duplicated:

   - The time of last data modification and time of last access
   - The user ID and group ID
   - The file mode

   If the user ID, group ID, or file mode of a regular file cannot be duplicated, the file mode bits S_ISUID and S_ISGID shall not be duplicated.

   When files are duplicated to another file system, the implementation may require that the process invoking *mv* has read access to each file being duplicated.

   If the duplication of the file hierarchy fails for any reason, *mv* shall write a diagnostic message to standard error, do nothing more with the current *source_file*, and go on to any remaining *source_file*s.

   If the duplication of the file characteristics fails for any reason, *mv* shall write a diagnostic message to standard error, but this failure shall not cause *mv* to modify its exit status.

6. The file hierarchy rooted in *source_file* shall be removed. If this fails for any reason, *mv* shall write a diagnostic message to the standard error, do nothing more with the current *source_file*, and go on to any remaining *source_file*s.

OPTIONS

The *mv* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–f Do not prompt for confirmation if the destination path exists. Any previous occurrences of the **–i** option is ignored.

–i Prompt for confirmation if the destination path exists. Any previous occurrences of the **–f** option is ignored.

Specifying more than one of the **–f** or **–i** options shall not be considered an error. The last option specified shall determine the behavior of *mv*.

OPERANDS

The following operands shall be supported:

source_file A pathname of a file or directory to be moved.

target_file A new pathname for the file or directory being moved.

target_dir A pathname of an existing directory into which to move the input files.

STDIN

The standard input shall be used to read an input line in response to each prompt specified in the STDERR section. Otherwise, the standard input shall not be used.

INPUT FILES

The input files specified by each *source_file* operand can be of any file type.

25198 **ENVIRONMENT VARIABLES**

25199 The following environment variables shall affect the execution of *mv*:

25200 *LANG* Provide a default value for the internationalization variables that are unset or null.
25201 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
25202 Internationalization Variables for the precedence of internationalization variables
25203 used to determine the values of locale categories.)

25204 *LC_ALL* If set to a non-empty string value, override the values of all the other
25205 internationalization variables.

25206 *LC_COLLATE*
25207 Determine the locale for the behavior of ranges, equivalence classes and multi-
25208 character collating elements used in the extended regular expression defined for
25209 the **yesexpr** locale keyword in the *LC_MESSAGES* category.

25210 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
25211 characters (for example, single-byte as opposed to multi-byte characters in
25212 arguments and input files), the behavior of character classes used in the extended
25213 regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES*
25214 category.

25215 *LC_MESSAGES*
25216 Determine the locale for the processing of affirmative responses that should be
25217 used to affect the format and contents of diagnostic messages written to standard
25218 error.

25219 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

25220 **ASYNCHRONOUS EVENTS**

25221 Default.

25222 **STDOUT**

25223 Not used.

25224 **STDERR**

25225 Prompts shall be written to the standard error under the conditions specified in the
25226 DESCRIPTION section. The prompts shall contain the *destination* pathname, but their format is
25227 otherwise unspecified. Otherwise, the standard error shall be used only for diagnostic messages.

25228 **OUTPUT FILES**

25229 The output files may be of any file type.

25230 **EXTENDED DESCRIPTION**

25231 None.

25232 **EXIT STATUS**

25233 The following exit values shall be returned:

25234 0 All input files were moved successfully.

25235 >0 An error occurred.

25236 **CONSEQUENCES OF ERRORS**

25237 If the copying or removal of *source_file* is prematurely terminated by a signal or error, *mv* may
25238 leave a partial copy of *source_file* at the source or destination. The *mv* utility shall not modify
25239 both *source_file* and the destination path simultaneously; termination at any point shall leave
25240 either *source_file* or the destination path complete.

APPLICATION USAGE

Some implementations mark for update the *st_ctime* field of renamed files and some do not. Applications which make use of the *st_ctime* field may behave differently with respect to renamed files unless they are designed to allow for either behavior.

EXAMPLES

If the current directory contains only files **a** (of any type defined by the System Interfaces volume of IEEE Std 1003.1-200x), **b** (also of any type), and a directory **c**:

```
mv a b c
mv c d
```

results with the original files **a** and **b** residing in the directory **d** in the current directory.

RATIONALE

Early proposals diverged from the SVID and BSD historical practice in that they required that when the destination path exists, the **–f** option is not specified, and input is not a terminal, *mv* fails. This was done for compatibility with *cp*. The current text returns to historical practice. It should be noted that this is consistent with the *rename*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x, which does not require write permission on the target.

For absolute clarity, paragraph (1), describing the behavior of *mv* when prompting for confirmation, should be interpreted in the following manner:

```
if (exists AND (NOT f_option) AND
    ((not_writable AND input_is_terminal) OR i_option))
```

The **–i** option exists on BSD systems, giving applications and users a way to avoid accidentally unlinking files when moving others. When the standard input is not a terminal, the 4.3 BSD *mv* deletes all existing destination paths without prompting, even when **–i** is specified; this is inconsistent with the behavior of the 4.3 BSD *cp* utility, which always generates an error when the file is unwritable and the standard input is not a terminal. The standard developers decided that use of **–i** is a request for interaction, so when the *destination* path exists, the utility takes instructions from whatever responds to standard input.

The *rename*( ) function is able to move directories within the same file system. Some historical versions of *mv* have been able to move directories, but not to a different file system. The standard developers considered that this was an annoying inconsistency, so this volume of IEEE Std 1003.1-200x requires directories to be able to be moved even across file systems. There is no **–R** option to confirm that moving a directory is actually intended, since such an option was not required for moving directories in historical practice. Requiring the application to specify it sometimes, depending on the destination, seemed just as inconsistent. The semantics of the *rename*( ) function were preserved as much as possible. For example, *mv* is not permitted to ''rename'' files to or from directories, even though they might be empty and removable.

Historic implementations of *mv* did not exit with a non-zero exit status if they were unable to duplicate any file characteristics when moving a file across file systems, nor did they write a diagnostic message for the user. The former behavior has been preserved to prevent scripts from breaking; a diagnostic message is now required, however, so that users are alerted that the file characteristics have changed.

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified because implementations may desire more descriptive prompts than those used on historical implementations. Therefore, an application not using the **–f** option or using the **–i** option relies on the system to provide the most suitable dialog directly with the user, based on the behavior specified.

When *mv* is dealing with a single file system and *source_file* is a symbolic link, the link itself is moved as a consequence of the dependence on the *rename*() functionality, per the DESCRIPTION. Across file systems, this has to be made explicit.

FUTURE DIRECTIONS

None.

SEE ALSO

cp, *ln*

CHANGE HISTORY

First released in Issue 2.

Issue 6

The *mv* utility is changed to describe processing of symbolic links as specified in the IEEE P1003.2b draft standard. |

The APPLICATION USAGE section is added. |

25301 **NAME**

25302 newgrp — change to a new group

25303 **SYNOPSIS**

25304 UP
```
newgrp [-l][group
```
25305

25306 **DESCRIPTION**

25307 The *newgrp* utility shall create a new shell execution environment with a new real and effective
25308 group identification. Of the attributes listed in Section 2.12 (on page 2263), the new shell
25309 execution environment shall retain the working directory, file creation mask, and exported
25310 variables from the previous environment (that is, open files, traps, unexported variables, alias
25311 definitions, shell functions, and *set* options may be lost). All other aspects of the process
25312 environment that are preserved by the *exec* family of functions defined in the System Interfaces
25313 volume of IEEE Std 1003.1-200x shall also be preserved by *newgrp*; whether other aspects are
25314 preserved is unspecified.

25315 A failure to assign the new group identifications (for example, for security or password-related
25316 reasons) shall not prevent the new shell execution environment from being created.

25317 The *newgrp* utility shall affect the supplemental groups for the process as follows:

25318 • On systems where the effective group ID is normally in the supplementary group list (or
25319 whenever the old effective group ID actually is in the supplementary group list):

25320 — If the new effective group ID is also in the supplementary group list, *newgrp* shall change
25321 the effective group ID.

25322 — If the new effective group ID is not in the supplementary group list, *newgrp* shall add the
25323 new effective group ID to the list, if there is room to add it.

25324 • On systems where the effective group ID is not normally in the supplementary group list (or
25325 whenever the old effective group ID is not in the supplementary group list):

25326 — If the new effective group ID is in the supplementary group list, *newgrp* shall delete it.

25327 — If the old effective group ID is not in the supplementary list, *newgrp* shall add it if there is
25328 room.

25329 **Note:** The System Interfaces volume of IEEE Std 1003.1-200x does not specify whether the effective
25330 group ID of a process is included in its supplementary group list.

25331 With no operands, *newgrp* shall change the effective group back to the groups identified in the
25332 user's user entry, and shall set the list of supplementary groups to that set in the user's group
25333 database entries.

25334 If a password is required for the specified group, and the user is not listed as a member of that
25335 group in the group database, the user shall be prompted to enter the correct password for that
25336 group. If the user is listed as a member of that group, no password shall be requested. If no |
25337 password is required for the specified group, it is implementation-defined whether users not |
25338 listed as members of that group can change to that group. Whether or not a password is
25339 required, implementation-defined system accounting or security mechanisms may impose
25340 additional authorization restrictions that may cause *newgrp* to write a diagnostic message and
25341 suppress the changing of the group identification.

25342 **OPTIONS**

25343 The *newgrp* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
25344 12.2, Utility Syntax Guidelines.

The following option shall be supported:

–l (The letter ell.) Change the environment to what would be expected if the user actually logged in again.

OPERANDS

The following operand shall be supported:

group A group name from the group database or a non-negative numeric group ID. Specifies the group ID to which the real and effective group IDs shall be set. If *group* is a non-negative numeric string and exists in the group database as a group name (see *getgrnam*( )), the numeric group ID associated with that group name shall be used as the group ID.

STDIN

Not used.

INPUT FILES

The file **/dev/tty** shall be used to read a single line of text for password checking, when one is required.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *newgrp*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used for diagnostic messages and a prompt string for a password, if one is required. Diagnostic messages may be written in cases where the exit status is not available. See the EXIT STATUS section.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

If *newgrp* succeeds in creating a new shell execution environment, whether or not the group identification was changed successfully, the exit status shall be the exit status of the shell. Otherwise, the following exit value shall be returned:

>0 An error occurred.

CONSEQUENCES OF ERRORS

The invoking shell may terminate.

APPLICATION USAGE

There is no convenient way to enter a password into the Group Database. Use of group passwords is not encouraged, because by their very nature they encourage poor security practices. Group passwords may disappear in the future.

A common implementation of *newgrp* is that the current shell uses *exec* to overlay itself with *newgrp*, which in turn overlays itself with a new shell after changing group. On some implementations, however, this may not occur and *newgrp* may be invoked as a subprocess.

The *newgrp* command is intended only for use from an interactive terminal. It does not offer a useful interface for the support of applications.

The exit status of *newgrp* is generally inapplicable. If *newgrp* is used in a script, in most cases it successfully invokes a new shell and the rest of the original shell script is bypassed when the new shell exits. Used interactively, *newgrp* displays diagnostic messages to indicate problems. But usage such as:

```
newgrp foo
echo $?
```

is not useful because the new shell might not have access to any status *newgrp* may have generated (and most historical systems do not provide this status). A zero status echoed here does not necessarily indicate that the user has changed to the new group successfully. Following *newgrp* with the *id* command provides a portable means of determining whether the group change was successful or not.

EXAMPLES

None.

RATIONALE

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 database, but how this would be implemented is not specified.

The *newgrp* utility was retained in this volume of IEEE Std 1003.1-200x, even given the existence of the multiple group permissions feature in the System Interfaces volume of IEEE Std 1003.1-200x, for several reasons. First, in some implementations, the group ownership of a newly created file is determined by the group of the directory in which the file is created, as allowed by the System Interfaces volume of IEEE Std 1003.1-200x; on other implementations, the group ownership of a newly created file is determined by the effective group ID. On implementations of the latter type, *newgrp* allows files to be created with a specific group ownership. Finally, many implementations use the real group ID in accounting, and on such systems, *newgrp* allows the accounting identity of the user to be changed.

25433 **FUTURE DIRECTIONS**

25434 None.

25435 **SEE ALSO**

25436 *sh*, the System Interfaces volume of IEEE Std 1003.1-200x, *exec*

25437 **CHANGE HISTORY**

25438 First released in Issue 2.

25439 **Issue 6**

25440 This utility is now marked as part of the User Portability Utilities option.

25441 The obsolescent SYNOPSIS is removed.

25442 The text describing supplemental groups is no longer conditional on {NGROUPS_MAX} being
25443 greater than 1. This is because {NGROUPS_MAX} now has a minimum value of 8. This is a FIPS
25444 requirement.

NAME

nice — invoke a utility with an altered nice value

SYNOPSIS

UP

```
nice [–n increment] utility [argument...]
```

DESCRIPTION

The *nice* utility shall invoke a utility, requesting that it be run with a different nice value (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.239, Nice Value). With no options and only if the user has appropriate privileges, the executed utility shall be run with a nice value that is some implementation-defined quantity less than or equal to the nice value of the current process. If the user lacks appropriate privileges to affect the nice value in the requested manner, the *nice* utility shall not affect the nice value; in this case, a warning message may be written to standard error, but this shall not prevent the invocation of *utility* or affect the exit status.

OPTIONS

The *nice* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following option is supported:

–n *increment* A positive or negative decimal integer which shall have the same effect on the execution of the utility as if the utility had called the *nice*( ) function with the numeric value of the *increment* option-argument.

OPERANDS

The following operands shall be supported:

utility The name of a utility that is to be invoked. If the *utility* operand names any of the special built-in utilities in Section 2.14 (on page 2266), the results are undefined.

argument Any string to be supplied as an argument when invoking the utility named by the *utility* operand.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *nice*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

25489 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

25490 *PATH* Determine the search path used to locate the utility to be invoked. See the Base
25491 Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

25492 **ASYNCHRONOUS EVENTS**
25493 Default.

25494 **STDOUT**
25495 Not used.

25496 **STDERR**
25497 The standard error shall be used only for diagnostic messages.

25498 **OUTPUT FILES**
25499 None.

25500 **EXTENDED DESCRIPTION**
25501 None.

25502 **EXIT STATUS**
25503 If the *utility* utility is invoked, the exit status of *nice* shall be the exit status of *utility*; otherwise,
25504 the *nice* utility shall exit with one of the following values:

25505 1-125 An error occurred in the *nice* utility.
25506 126 The utility specified by *utility* was found but could not be invoked.
25507 127 The utility specified by *utility* could not be found.

25508 **CONSEQUENCES OF ERRORS**
25509 Default.

25510 **APPLICATION USAGE**
25511 The only guaranteed portable uses of this utility are:

25512 *nice utility*
25513 Run *utility* with the default lower nice value.

25514 *nice* **–n** <*positive integer*> *utility*
25515 Run *utility* with a lower nice value.

25516 On some implementations they have no discernible effect on the invoked utility and on some
25517 others they are exactly equivalent.

25518 Historical systems have frequently supported the <*positive integer*> up to 20. Since there is no
25519 error penalty associated with guessing a number that is too high, users without access to the
25520 system conformance document (to see what limits are actually in place) could use the historical
25521 1 to 20 range or attempt to use very large numbers if the job should be truly low priority.

25522 The nice value value of a process can be displayed using the command:

25523 `ps –o nice`

25524 The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if
25525 an error occurs so that applications can distinguish ‘‘failure to find a utility’’ from ‘‘invoked
25526 utility exited with an error indication’’. The value 127 was chosen because it is not commonly
25527 used for other meanings; most utilities use small values for ‘‘normal error conditions’’ and the
25528 values above 128 can be confused with termination due to receipt of a signal. The value 126 was
25529 chosen in a similar manner to indicate that the utility could be found, but not invoked. Some
25530 scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction
25531 between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to
25532 *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for

any other reason.

EXAMPLES

None.

RATIONALE

Due to the text about the limits of the nice value being implementation-defined, *nice* is not actually required to change the nice value 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.

The 4.3 BSD 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 results 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 4.3 BSD 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 in this volume of IEEE Std 1003.1-200x.

The term ''utility'' is used, rather than ''command'', to highlight the fact that shell compound commands, pipelines, and so on, 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 volume of IEEE Std 1003.1-200x.

Historical implementations of *nice* provide a nice value range of 40 or 41 discrete steps, with the default nice value being the midpoint of that range. By default, they lower the nice value 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 nice value, which is not always the default.

The definition of nice value is not intended to suggest that all processes in a system have priorities that are comparable. Scheduling policy extensions such as the realtime priorities in the System Interfaces volume of IEEE Std 1003.1-200x make the notion of a single underlying priority for all scheduling policies problematic. Some implementations may implement the *nice*-related features to affect all processes on the system, others to affect just the general time-sharing activities implied by this volume of IEEE Std 1003.1-200x, and others may have no effect at all. Because of the use of ''implementation-defined'' in *nice* and *renice*, a wide range of implementation strategies are possible.

FUTURE DIRECTIONS

None.

SEE ALSO

renice, the System Interfaces volume of IEEE Std 1003.1-200x, *nice*( )

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The obsolescent SYNOPSIS is removed.

25577 **NAME**

25578 nl — line numbering filter

25579 **SYNOPSIS**

25580 XSI
```
nl [–p][–b type][–d delim][–f type][–h type][–i incr][–l num][–n format]
    [–s sep][–v startnum][–w width][file]
```
25581

25582

25583 **DESCRIPTION**

25584 The *nl* utility shall read lines from the named *file* or the standard input if no *file* is named and
25585 shall reproduce the lines to standard output. Lines shall be numbered on the left. Additional
25586 functionality may be provided in accordance with the command options in effect.

25587 The *nl* utility views the text it reads in terms of logical pages. Line numbering shall be reset at
25588 the start of each logical page. A logical page consists of a header, a body, and a footer section.
25589 Empty sections are valid. Different line numbering options are independently available for
25590 header, body, and footer (for example, no numbering of header and footer lines while
25591 numbering blank lines only in the body).

25592 The starts of logical page sections shall be signaled by input lines containing nothing but the
25593 following delimiter characters:

25594

Line	Start of
\:\:\:	Header
\:\:	Body
\:	Footer

25595
25596
25597

25598 Unless otherwise specified, *nl* shall assume the text being read is in a single logical page body.

25599 **OPTIONS**

25600 The *nl* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
25601 Utility Syntax Guidelines. Only one file can be named.

25602 The following options shall be supported:

25603 **–b** *type* Specify which logical page body lines shall be numbered. Recognized *types* and
25604 their meaning are:

25605 **a** Number all lines.

25606 **t** Number only non-empty lines.

25607 **n** No line numbering.

25608 **p***string* Number only lines that contain the basic regular expression specified in
25609 *string*.

25610 The default *type* for logical page body shall be **t** (text lines numbered).

25611 **–d** *delim* Specify the delimiter characters that indicate the start of a logical page section.
25612 These can be changed from the default characters "\:" to two user-specified
25613 characters. If only one character is entered, the second character shall remain the
25614 default character ':'.

25615 **–f** *type* Specify the same as **b** *type* except for footer. The default for logical page footer
25616 shall be **n** (no lines numbered).

25617 **–h** *type* Specify the same as **b** *type* except for header. The default *type* for logical page
25618 header shall be **n** (no lines numbered).

25619 **–i** *incr* Specify the increment value used to number logical page lines. The default shall be
25620 1.

25621 **–l** *num* Specify the number of blank lines to be considered as one. For example, **–l 2** results
25622 in only the second adjacent blank line being numbered (if the appropriate **–h a**,
25623 **–b a**, or **–f a** option is set). The default shall be 1.

25624 **–n** *format* Specify the line numbering format. Recognized values are: **ln**, left justified, leading
25625 zeros suppressed; **rn**, right justified, leading zeros suppressed; **rz**, right justified,
25626 leading zeros kept. The default *format* shall be **rn** (right justified).

25627 **–p** Specify that numbering should not be restarted at logical page delimiters.

25628 **–s** *sep* Specify the characters used in separating the line number and the corresponding
25629 text line. The default *sep* shall be a <tab>.

25630 **–v** *startnum* Specify the initial value used to number logical page lines. The default shall be 1.

25631 **–w** *width* Specify the number of characters to be used for the line number. The default *width*
25632 shall be 6.

25633 **OPERANDS**

25634 The following operand shall be supported:

25635 *file* A pathname of a text file to be line-numbered.

25636 **STDIN**

25637 The standard input is a text file that is used if no *file* operand is given.

25638 **INPUT FILES**

25639 The input file named by the *file* operand is a text file.

25640 **ENVIRONMENT VARIABLES**

25641 The following environment variables shall affect the execution of *nl*:

25642 *LANG* Provide a default value for the internationalization variables that are unset or null.
25643 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
25644 Internationalization Variables for the precedence of internationalization variables
25645 used to determine the values of locale categories.)

25646 *LC_ALL* If set to a non-empty string value, override the values of all the other
25647 internationalization variables.

25648 *LC_COLLATE*
25649 Determine the locale for the behavior of ranges, equivalence classes and multi-
25650 character collating elements within regular expressions.

25651 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
25652 characters (for example, single-byte as opposed to multi-byte characters in
25653 arguments and input files), the behavior of character classes within regular
25654 expressions, and for deciding which characters are in character class **graph** (for the
25655 **–b t**, **–f t**, and **–h t** options).

25656 *LC_MESSAGES*
25657 Determine the locale that should be used to affect the format and contents of
25658 diagnostic messages written to standard error.

25659 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

25660 **ASYNCHRONOUS EVENTS**
25661 Default.

25662 **STDOUT**
25663 The standard output shall be a text file in the following format:

25664 `"%s%s%s", <line number>, <separator>, <input line>`

25665 where <*line number*> is one of the following numeric formats:

25666 `%6d` When the **rn** format is used (the default; see **–n**).

25667 `%06d` When the **rz** format is used.

25668 `%–6d` When the *ln* format is used.

25669 <empty> When line numbers are suppressed for a portion of the page; the <*separator*> is also
25670 suppressed.

25671 In the preceding list, the number 6 is the default width; the **–w** option can change this value.

25672 **STDERR**
25673 The standard error shall be used only for diagnostic messages.

25674 **OUTPUT FILES**
25675 None.

25676 **EXTENDED DESCRIPTION**
25677 None.

25678 **EXIT STATUS**
25679 The following exit values shall be returned:

25680 0 Successful completion.

25681 >0 An error occurred.

25682 **CONSEQUENCES OF ERRORS**
25683 Default.

25684 **APPLICATION USAGE**
25685 In using the **–d** *delim* option, care should be taken to escape characters that have special meaning
25686 to the command interpreter.

25687 **EXAMPLES**
25688 The command:

25689 `nl –v 10 –i 10 –d \!+ file1`

25690 numbers *file1* starting at line number 10 with an increment of 10. The logical page delimiter is
25691 `"!+"`. Note that the `'!'` has to be escaped when using *csh* as a command interpreter because of
25692 its history substitution syntax. For *ksh* and *sh* the escape is not necessary, but does not do any
25693 harm.

25694 **RATIONALE**
25695 None.

25696 **FUTURE DIRECTIONS**
25697 None.

SEE ALSO

pr

CHANGE HISTORY

First released in Issue 2.

Issue 5

The option [**–f** *type*] is added to the SYNOPSIS. The option descriptions are presented in alphabetic order. The description of **–bt** is changed to ‘‘Number only non-empty lines’’.

Issue 6

The obsolescent behavior allowing the options to be intermingled with the optional *file* operand is removed.

25708 **NAME**

25709 nm — write the name list of an object file (**DEVELOPMENT**)

25710 **SYNOPSIS**

25711 UP SD XSI `nm [–APv][–efox][ –g| –u][–t format] file...`

25712

25713 **DESCRIPTION**

25714 This utility shall be provided on systems that support both the User Portability Utilities option
25715 and the Software Development Utilities option. On other systems it is optional. Certain options
25716 are only available on XSI-conformant systems.

25717 The *nm* utility shall display symbolic information appearing in the object file, executable file or
25718 object-file library named by *file*. If no symbolic information is available for a valid input file, the
25719 *nm* utility shall report that fact, but not consider it an error condition.

25720 XSI The default base used when numeric values are written is unspecified. On XSI-conformant
25721 systems, it shall be decimal.

25722 **OPTIONS**

25723 The *nm* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
25724 12.2, Utility Syntax Guidelines.

25725 The following options shall be supported:

25726 **–A** Write the full pathname or library name of an object on each line.

25727 XSI **–e** Write only external (global) and static symbol information.

25728 XSI **–f** Produce full output. Write redundant symbols (**.text**, **.data**, and **.bss**), normally
25729 suppressed.

25730 **–g** Write only external (global) symbol information.

25731 XSI **–o** Write numeric values in octal (equivalent to **–t o**).

25732 **–P** Write information in a portable output format, as specified in the STDOUT section.

25733 **–t** *format* Write each numeric value in the specified format. The format shall be dependent
25734 on the single character used as the *format* option-argument:

25735 XSI d The offset is written in decimal (default).

25736 o The offset is written in octal.

25737 x The offset is written in hexadecimal.

25738 **–u** Write only undefined symbols.

25739 **–v** Sort output by value instead of alphabetically.

25740 XSI **–x** Write numeric values in hexadecimal (equivalent to **–t x**).

25741 **OPERANDS**

25742 The following operand shall be supported:

25743 *file* A pathname of an object file, executable file, or object-file library.

25744 **STDIN**

25745 See the INPUT FILES section.

25746 **INPUT FILES**

25747 The input file shall be an object file, an object-file library whose format is the same as those
25748 produced by the *ar* utility for link editing, or an executable file. The *nm* utility may accept
25749 additional implementation-defined object library formats for the input file.

25750 **ENVIRONMENT VARIABLES**

25751 The following environment variables shall affect the execution of *nm*:

25752 *LANG* Provide a default value for the internationalization variables that are unset or null.
25753 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
25754 Internationalization Variables for the precedence of internationalization variables
25755 used to determine the values of locale categories.)

25756 *LC_ALL* If set to a non-empty string value, override the values of all the other
25757 internationalization variables.

25758 *LC_COLLATE*
25759 Determine the locale for character collation information for the symbol-name and
25760 symbol-value collation sequences.

25761 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
25762 characters (for example, single-byte as opposed to multi-byte characters in
25763 arguments).

25764 *LC_MESSAGES*
25765 Determine the locale that should be used to affect the format and contents of
25766 diagnostic messages written to standard error.

25767 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

25768 **ASYNCHRONOUS EVENTS**

25769 Default.

25770 **STDOUT**

25771 If symbolic information is present in the input files, then for each file or for each member of an
25772 archive, the *nm* utility shall write the following information to standard output. By default, the
25773 format is unspecified, but the output shall be sorted alphabetically by symbol name:

25774 • Library or object name, if **–A** is specified

25775 • Symbol name

25776 • Symbol type, which shall either be one of the following single characters or an
25777 implementation-defined type represented by a single character:

25778 A Global absolute symbol.

25779 a Local absolute symbol.

25780 B Global ‘‘bss’’ (that is, uninitialized data space) symbol.

25781 b Local bss symbol.

25782 D Global data symbol.

25783 d Local data symbol.

25784 T Global text symbol.

25785 t Local text symbol.

25786 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> shall be formatted as follows:

- If **–A** is not specified, *<library/object name>* shall be an empty string.
- If **–A** is specified and the corresponding *file* operand does not name a library:

  ```
  "%s: ", <file>
  ```

- If **–A** is specified and the corresponding *file* operand names a library. In this case, *<object file>* shall name the object file in the library containing the symbol being described:

  ```
  "%s[%s]: ", <file>, <object file>
  ```

If **–A** is not specified, then if more than one *file* operand is specified or if only one *file* operand is specified and it names a library, *nm* shall write a line identifying the object containing the following symbols before the lines containing those symbols, in the form:

- If the corresponding *file* operand does not name a library:

  ```
  "%s:\n", <file>
  ```

- If the corresponding *file* operand names a library; in this case, *<object file>* shall be the name of the file in the library containing the following symbols:

  ```
  "%s[%s]:\n", <file>, <object file>
  ```

If **–P** is specified, but **–t** is not, the format shall be as if **–t x** had been specified.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Mechanisms for dynamic linking make this utility less meaningful when applied to an executable file because a dynamically linked executable may omit numerous library routines that would be found in a statically linked executable.

EXAMPLES

None.

RATIONALE

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.

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 confusion with the **–o** from System V (which has been provided here as **–t** and as **–o** on XSI-conformant systems).

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.

It was recognized that mechanisms for dynamic linking make this utility less meaningful when applied to an executable file (because a dynamically linked executable file may omit numerous library routines that would be found in a statically linked executable file), but the value of *nm* during software development was judged to outweigh other limitations.

The default output format of *nm* is not specified because of differences in historical implementations. The **–P** option was added to allow some type of portable output format. After a comparison of the different formats used in SunOS, BSD, SVR3, and SVR4, it was decided to create one that did not match the current format of any of these four systems. The format devised is easy to parse by humans, easy to parse in shell scripts, and does not need to vary depending on locale (because no English descriptions are included). All of the systems currently have the information available to use this format.

The format given in *nm* STDOUT uses spaces between the fields, which may be any number of <blank>s required to align the columns. The single-character types were selected to match historical practice, and the requirement that implementation additions also be single characters made parsing the information easier for shell scripts.

FUTURE DIRECTIONS

None.

SEE ALSO

ar, *c99*

CHANGE HISTORY

First released in Issue 2.

25868 **Issue 6**

25869 This utility is now marked as supported when both the User Portability Utilities option and the
25870 Software Development Utilities option are supported.

25871 **NAME**

25872 nohup — invoke a utility immune to hangups

25873 **SYNOPSIS**

25874 nohup *utility* **[***argument...***]**

25875 **DESCRIPTION**

25876 The *nohup* utility shall invoke the utility named by the *utility* operand with arguments supplied
25877 as the *argument* operands. At the time the named *utility* is invoked, the SIGHUP signal shall be
25878 set to be ignored.

25879 If the standard output is a terminal, all output written by the named *utility* to its standard output
25880 shall be appended to the end of the file **nohup.out** in the current directory. If **nohup.out** cannot
25881 be created or opened for appending, the output shall be appended to the end of the file
25882 **nohup.out** in the directory specified by the *HOME* environment variable. If neither file can be
25883 created or opened for appending, *utility* shall not be invoked. If a file is created, the file's
25884 permission bits shall be set to S_IRUSR | S_IWUSR.

25885 If the standard error is a terminal, all output written by the named *utility* to its standard error
25886 shall be redirected to the same file descriptor as the standard output.

25887 **OPTIONS**

25888 None.

25889 **OPERANDS**

25890 The following operands shall be supported:

25891 *utility* The name of a utility that is to be invoked. If the *utility* operand names any of the
25892 special built-in utilities in Section 2.14 (on page 2266), the results are undefined.

25893 *argument* Any string to be supplied as an argument when invoking the utility named by the
25894 *utility* operand.

25895 **STDIN**

25896 Not used.

25897 **INPUT FILES**

25898 None.

25899 **ENVIRONMENT VARIABLES**

25900 The following environment variables shall affect the execution of *nohup*:

25901 *HOME* Determine the pathname of the user's home directory: if the output file **nohup.out**
25902 cannot be created in the current directory, the *nohup* utility shall use the directory
25903 named by *HOME* to create the file.

25904 *LANG* Provide a default value for the internationalization variables that are unset or null.
25905 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
25906 Internationalization Variables for the precedence of internationalization variables
25907 used to determine the values of locale categories.)

25908 *LC_ALL* If set to a non-empty string value, override the values of all the other
25909 internationalization variables.

25910 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
25911 characters (for example, single-byte as opposed to multi-byte characters in
25912 arguments).

25913 *LC_MESSAGES*

25914 Determine the locale that should be used to affect the format and contents of

diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

PATH Determine the search path that is used to locate the utility to be invoked. See the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

ASYNCHRONOUS EVENTS

The *nohup* utility shall take the standard action for all signals except that SIGHUP shall be ignored.

STDOUT

If the standard output is not a terminal, the standard output of *nohup* shall be the standard output generated by the execution of the *utility* specified by the operands. Otherwise, nothing shall be written to the standard output.

STDERR

If the standard output is a terminal, a message shall be written to the standard error, indicating the name of the file to which the output is being appended. The name of the file shall be either **nohup.out** or **$HOME/nohup.out**.

OUTPUT FILES

If the standard output is a terminal, all output written by the named *utility* to the standard output and standard error is appended to the file **nohup.out**, which is created if it does not already exist.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

126 The utility specified by *utility* was found but could not be invoked.

127 An error occurred in the *nohup* utility or the utility specified by *utility* could not be found.

Otherwise, the exit status of *nohup* shall be that of the utility specified by the *utility* operand.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if an error occurs so that applications can distinguish ‘‘failure to find a utility’’ from ‘‘invoked utility exited with an error indication’’. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for ‘‘normal error conditions’’ and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for any other reason.

EXAMPLES

It is frequently desirable to apply *nohup* 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 *nohup* applies to everything in the file.

Alternatively, the following command can be used to apply *nohup* to a complex command:

```
nohup sh –c 'complex-command-line'
```

RATIONALE

The 4.3 BSD version ignores SIGTERM and SIGHUP, and if **./nohup.out** cannot be used, it fails instead of trying to use **$HOME/nohup.out**.

The *csh* utility has a built-in version of *nohup* that acts differently from the POSIX Shell and Utilities *nohup*.

The term *utility* is used, rather than *command*, to highlight the fact that shell compound commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility* includes user application programs and shell scripts, not just the standard utilities.

Historical versions of the *nohup* utility use default file creation semantics. Some more recent versions use the permissions specified here as an added security precaution.

Some historical implementations ignore SIGQUIT in addition to SIGHUP; others ignore SIGTERM. An early proposal allowed, but did not require, SIGQUIT to be ignored. Several reviewers objected that *nohup* should only modify the handling of SIGHUP as required by this volume of IEEE Std 1003.1-200x.

FUTURE DIRECTIONS

None.

SEE ALSO

sh, the System Interfaces volume of IEEE Std 1003.1-200x, *signal*( )

CHANGE HISTORY

First released in Issue 2.

25982 **NAME**

25983 od — dump files in various formats

25984 **SYNOPSIS**

25985 `od [–v][–A address_base][–j skip][–N count][–t type_string]...`
25986 `[file...]`

25987 XSI `od [–bcdosx][file] [[+]offset[.][b]]`
25988

25989 **DESCRIPTION**

25990 The *od* utility shall write the contents of its input files to standard output in a user-specified
25991 format.

25992 **OPTIONS**

25993 The *od* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
25994 XSI Utility Syntax Guidelines, except that the order of presentation of the **–t** options and the
25995 **–bcdosx** options is significant.

25996 The following options shall be supported:

25997 **–A** *address_base*
25998 Specify the input offset base. See the EXTENDED DESCRIPTION section. The
25999 application shall ensure that the *address_base* option-argument is a character. The
26000 characters `'d'`, `'o'`, and `'x'` specify that the offset base shall be written in
26001 decimal, octal, or hexadecimal, respectively. The character `'n'` specifies that the
26002 offset shall not be written.

26003 XSI **–b** Interpret bytes in octal. This shall be equivalent to **–t o1**. |

26004 XSI **–c** Interpret bytes as characters specified by the current setting of the *LC_CTYPE*
26005 category. Certain non-graphic characters appear as C escapes: `"NUL=\0"`,
26006 `"BS=\b"`, `"FF=\f"`, `"NL=\n"`, `"CR=\r"`, `"HT=\t"`; others appear as 3-digit octal
26007 numbers.

26008 XSI **–d** Interpret *word*s (two-byte units) in unsigned decimal. This shall be equivalent to |
26009 **–t u2**.

26010 **–j** *skip* Jump over *skip* bytes from the beginning of the input. The *od* utility shall read or
26011 seek past the first *skip* bytes in the concatenated input files. If the combined input
26012 is not at least *skip* bytes long, the *od* utility shall write a diagnostic message to
26013 standard error and exit with a non-zero exit status.

26014 By default, the *skip* option-argument shall be interpreted as a decimal number.
26015 With a leading 0x or 0X, the offset shall be interpreted as a hexadecimal number;
26016 otherwise, with a leading `'0'`, the offset shall be interpreted as an octal number.
26017 Appending the character `'b'`, `'k'`, or `'m'` to offset shall cause it to be interpreted
26018 as a multiple of 512, 1 024, or 1 048 576 bytes, respectively. If the *skip* number is
26019 hexadecimal, any appended `'b'` shall be considered to be the final hexadecimal
26020 digit.

26021 **–N** *count* Format no more than *count* bytes of input. By default, *count* shall be interpreted as
26022 a decimal number. With a leading 0x or 0X, *count* shall be interpreted as a
26023 hexadecimal number; otherwise, with a leading `'0'`, it shall be interpreted as an
26024 octal number. If *count* bytes of input (after successfully skipping, if **–j** *skip* is
26025 specified) are not available, it shall not be considered an error; the *od* utility shall
26026 format the input that is available.

26027 XSI **–o** Interpret *word*s (two-byte units) in octal. This shall be equivalent to **–t o2**.

26028 XSI **–s** Interpret *word*s (two-byte units) in signed decimal. This shall be equivalent to
26029 **–t d2**.

26030 **–t** *type_string*
26031 Specify one or more output types. See the EXTENDED DESCRIPTION section. The
26032 application shall ensure that the *type_string* option-argument is a string specifying
26033 the types to be used when writing the input data. The string shall consist of the
26034 type specification characters `a`, `c`, `d`, `f`, `o`, `u`, and `x`, specifying named character,
26035 character, signed decimal, floating point, octal, unsigned decimal, and
26036 hexadecimal, respectively. The type specification characters `d`, `f`, `o`, `u`, and `x` can be
26037 followed by an optional unsigned decimal integer that specifies the number of
26038 bytes to be transformed by each instance of the output type. The type specification
26039 character `f` can be followed by an optional `F`, `D`, or `L` indicating that the conversion
26040 should be applied to an item of type **float**, **double**, or **long double**, respectively.
26041 The type specification characters `d`, `o`, `u` and `x` can be followed by an optional `C`, `S`,
26042 `I`, or `L` indicating that the conversion should be applied to an item of type **char**,
26043 **short**, **int**, or **long**, respectively. Multiple types can be concatenated within the
26044 same *type_string* and multiple **–t** options can be specified. Output lines shall be
26045 written for each type specified in the order in which the type specification
26046 characters are specified.

26047 **–v** Write all input data. Without the **–v** option, any number of groups of output lines,
26048 which would be identical to the immediately preceding group of output lines
26049 (except for the byte offsets), shall be replaced with a line containing only an
26050 asterisk (`'*'`).

26051 XSI **–x** Interpret *word*s (two-byte units) in hexadecimal. This shall be equivalent to **–t x2**.

26052 XSI Multiple types can be specified by using multiple **–bcdostx** options. Output lines are written for
26053 each type specified in the order in which the types are specified.

26054 **OPERANDS**
26055 The following operands shall be supported:

26056 *file* A pathname of a file to be read. If no *file* operands are specified, the standard input
26057 shall be used.

26058 If there are no more than two operands, none of the **–A**, **–j**, **–N**, or **–t** options is
26059 specified, and either of the following is true: the first character of the last operand
26060 is a plus sign (`'+'`), or there are two operands and the first character of the last
26061 XSI operand is numeric; the last operand shall be interpreted as an offset operand on
26062 XSI-conformant systems. Under these conditions, the results are unspecified on
26063 systems that are not XSI-conformant systems.

26064 XSI **[+]***offset***[.][b]** The *offset* operand specifies the offset in the file where dumping is to commence.
26065 This operand is normally interpreted as octal bytes. If `'.'` is appended, the offset
26066 shall be interpreted in decimal. If `'b'` is appended, the offset shall be interpreted
26067 in units of 512 bytes.

26068 **STDIN**
26069 The standard input shall be used only if no *file* operands are specified. See the INPUT FILES
26070 section.

26071 **INPUT FILES**

26072 The input files can be any file type.

26073 **ENVIRONMENT VARIABLES**

26074 The following environment variables shall affect the execution of *od*:

26075 *LANG* Provide a default value for the internationalization variables that are unset or null.
26076 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
26077 Internationalization Variables for the precedence of internationalization variables
26078 used to determine the values of locale categories.)

26079 *LC_ALL* If set to a non-empty string value, override the values of all the other
26080 internationalization variables.

26081 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
26082 characters (for example, single-byte as opposed to multi-byte characters in
26083 arguments and input files).

26084 *LC_MESSAGES*
26085 Determine the locale that should be used to affect the format and contents of
26086 diagnostic messages written to standard error.

26087 *LC_NUMERIC*
26088 Determine the locale for selecting the radix character used when writing floating-
26089 point formatted output.

26090 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

26091 **ASYNCHRONOUS EVENTS**

26092 Default.

26093 **STDOUT**

26094 See the EXTENDED DESCRIPTION section.

26095 **STDERR**

26096 The standard error shall be used only for diagnostic messages.

26097 **OUTPUT FILES**

26098 None.

26099 **EXTENDED DESCRIPTION**

26100 The *od* utility shall copy sequentially each input file to standard output, transforming the input
26101 XSI data according to the output types specified by the **–t** options or the **–bcdosx** options. If no
26102 output type is specified, the default output shall be as if **–t oS** had been specified.

26103 The number of bytes transformed by the output type specifier `c` may be variable depending on
26104 the *LC_CTYPE* category.

26105 The default number of bytes transformed by output type specifiers `d`, `f`, `o`, `u`, and `x` corresponds
26106 to the various C-language types as follows. If the *c99* compiler is present on the system, these
26107 specifiers shall correspond to the sizes used by default in that compiler. Otherwise, these sizes
26108 may vary among systems that conform to IEEE Std 1003.1-200x.

26109 • For the type specifier characters `d`, `o`, `u`, and `x`, the default number of bytes shall correspond
26110 to the size of the underlying implementation's basic integer type. For these specifier
26111 characters, the implementation shall support values of the optional number of bytes to be
26112 converted corresponding to the number of bytes in the C-language types **char**, **short**, **int**, and
26113 **long**. These numbers can also be specified by an application as the characters `'C'`, `'S'`, `'I'`,
26114 and `'L'`, respectively. The implementation shall also support the values 1, 2, 4, and 8, even if
26115 it provides no C-Language types of those sizes. The implementation shall support the

decimal value corresponding to the C-language type **long long**. The byte order used when interpreting numeric values is implementation-defined, but shall correspond to the order in which a constant of the corresponding type is stored in memory on the system.

- For the type specifier character `f`, the default number of bytes shall correspond to the number of bytes in the underlying implementation's basic double precision floating-point data type. The implementation shall support values of the optional number of bytes to be converted corresponding to the number of bytes in the C-language types **float, double**, and **long double**. These numbers can also be specified by an application as the characters `'F'`, `'D'`, and `'L'`, respectively.

The type specifier character `a` specifies that bytes shall be interpreted as named characters from the International Reference Version (IRV) of the ISO/IEC 646:1991 standard. Only the least significant seven bits of each byte shall be used for this type specification. Bytes with the values listed in the following table shall be written using the corresponding names for those characters.

Table 4-12 Named Characters in *od*

Value	Name	Value	Name	Value	Name	Value	Name
\000	**nul**	\001	**soh**	\002	**stx**	\003	**etx**
\004	**eot**	\005	**enq**	\006	**ack**	\007	**bel**
\010	**bs**	\011	**ht**	\012	**lf** or **nl***	\013	**vt**
\014	**ff**	\015	**cr**	\016	**so**	\017	**si**
\020	**dle**	\021	**dc1**	\022	**dc2**	\023	**dc3**
\024	**dc4**	\025	**nak**	\026	**syn**	\027	**etb**
\030	**can**	\031	**em**	\032	**sub**	\033	**esc**
\034	**fs**	\035	**gs**	\036	**rs**	\037	**us**
\040	**sp**	\177	**del**				

Note: The `"\012"` value may be written either as **lf** or **nl**.

The type specifier character `c` specifies that bytes shall be interpreted as characters specified by the current setting of the *LC_CTYPE* locale category. Characters listed in the table in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`) shall be written as the corresponding escape sequences, except that backslash shall be written as a single backslash and a NUL shall be written as `'\0'`. Other non-printable characters shall be written as one three-digit octal number for each byte in the character. If the size of a byte on the system is greater than nine bits, the format used for non-printable characters is implementation-defined. Printable multi-byte characters shall be written in the area corresponding to the first byte of the character; the two-character sequence `"**"` shall be written in the area corresponding to each remaining byte in the character, as an indication that the character is continued. When either the **–j** *skip* or **–N** *count* option is specified along with the `c` type specifier, and this results in an attempt to start or finish in the middle of a multi-byte character, the result is implementation-defined.

The input data shall be manipulated in blocks, where a block is defined as a multiple of the least common multiple of the number of bytes transformed by the specified output types. If the least common multiple is greater than 16, the results are unspecified. Each input block shall be written as transformed by each output type, one per written line, in the order that the output types were specified. If the input block size is larger than the number of bytes transformed by the output type, the output type shall sequentially transform the parts of the input block, and the output from each of the transformations shall be separated by one or more <blank>s.

If, as a result of the specification of the **–N** option or end-of-file being reached on the last input file, input data only partially satisfies an output type, the input shall be extended sufficiently with null bytes to write the last byte of the input.

Unless **–A n** is specified, the first output line produced for each input block shall be preceded by the input offset, cumulative across input files, of the next byte to be written. The format of the input offset is unspecified; however, it shall not contain any <blank>s, shall start at the first character of the output line, and shall be followed by one or more <blank>s. In addition, the offset of the byte following the last byte written shall be written after all the input data has been processed, but shall not be followed by any <blank>s.

If no **–A** option is specified, the input offset base is unspecified.

EXIT STATUS

The following exit values shall be returned:

0 All input files were processed successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

XSI-conformant applications are warned not to use filenames starting with `'+'` or a first operand starting with a numeric character so that the old functionality can be maintained by implementations, unless they specify one of the **–A**, **–j**, or **–N** options. To guarantee that one of these filenames is always interpreted as a filename, an application could always specify the address base format with the **–A** option.

EXAMPLES

If a file containing 128 bytes with decimal values zero to 127, in increasing order, is supplied as standard input to the command:

```
od –A d –t a
```

on an implementation using an input block size of 16 bytes, the standard output, independent of the current locale setting, would be similar to:

```
0000000 nul soh stx etx eot enq ack bel  bs  ht  nl  vt  ff  cr  so  si
0000016 dle dc1 dc2 dc3 dc4 nak syn etb can  em sub esc  fs  gs  rs  us
0000032  sp   !   "   #   $   %   &   ’   (   )   *   +   ,   –   .   /
0000048   0   1   2   3   4   5   6   7   8   9   :   ;   <   =   >   ?
0000064   @   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O
0000080   P   Q   R   S   T   U   V   W   X   Y   Z   [   \   ]   ^   _
0000096   ‘   a   b   c   d   e   f   g   h   i   j   k   l   m   n   o
0000112   p   q   r   s   t   u   v   w   x   y   z   {   |   }   ~ del
0000128
```

Note that this volume of IEEE Std 1003.1-200x allows **nl** or **lf** to be used as the name for the ISO / IEC 646: 1991 standard IRV character with decimal value 10. The IRV names this character **lf** (line feed), but traditional implementations have referred to this character as newline (**nl**) and the POSIX locale character set symbolic name for the corresponding character is a <newline>.

The command:

```
od –A o –t o2x2x –n 18
```

on a system with 32-bit words and an implementation using an input block size of 16 bytes could write 18 bytes in approximately the following format:

```
0000000 032056 031440 041123 042040 052516 044530 020043 031464
          342e   3320   4253   4420   554e   4958   2023   3334
             342e3320       42534420       554e4958       20233334
0000020 032472
          353a
             353a0000
0000022
```

The command:

```
od –A d –t f –t o4 –t x4 –n 24 –j 0x15
```

on a system with 64-bit doubles (for example, IEEE Std 754-1985 double precision floating-point format) would skip 21 bytes of input data and then write 24 bytes in approximately the following format:

```
0000000    1.00000000000000e+00    1.57350000000000e+01
         07774000000 00000000000 10013674121 35341217270
            3ff00000    00000000    402f3851    eb851eb8
0000016    1.40668230000000e+02
         10030312542 04370303230
            40619562    23e18698
0000024
```

RATIONALE

The *od* utility went through several names in early proposals, including *hd*, *xd*, and most recently *hexdump*. There were several objections to all of these based on the following reasons:

- The *hd* and *xd* names conflicted with historical utilities that behaved differently.
- The *hexdump* description was much more complex than needed for a simple dump utility.
- The *od* utility has been available on all historical implementations and there was no need to create a new name for a utility so similar to the historical *od* utility.

The original reasons for not standardizing historical *od* were also fairly widespread. Those reasons are given below along with rationale explaining why the standard developers believe that this version does not suffer from the indicated problem:

- The BSD and System V versions of *od* have diverged, and the intersection of features provided by both does not meet the needs of the user community. In fact, the System V version only provides a mechanism for dumping octal bytes and **short**s, signed and unsigned decimal **short**s, hexadecimal **short**s, and ASCII characters. BSD added the ability to dump **float**s, **double**s, named ASCII characters, and octal, signed decimal, unsigned decimal, and hexadecimal **long**s. The version presented here provides more normalized forms for dumping bytes, **short**s, **int**s, and **long**s in octal, signed decimal, unsigned decimal, and hexadecimal; **float**, **double**, and **long double**; and named ASCII as well as current locale characters.
- It would not be possible to come up with a compatible superset of the BSD and System V flags that met the requirements of the standard developers. The historical default *od* output is the specified default output of this utility. None of the option letters chosen for this version of *od* conflict with any of the options to historical versions of *od*.
- On systems with different sizes for **short**, **int**, and **long**, there was no way to ask for dumps of **int**s, even in the BSD version. Because of the way options are named, the name space could not be extended to solve these problems. This is why the **–t** option was added (with type specifiers more closely matched to the *printf*( ) formats used in the rest of this volume of

IEEE Std 1003.1-200x) and the optional field sizes were added to the `d`, `f`, `o`, `u`, and `x` type specifiers. It is also one of the reasons why the historical practice was not mandated as a required obsolescent form of *od*. (Although the old versions of *od* are not listed as an obsolescent form, implementations are urged to continue to recognize the older forms for several more years.) The `a`, `c`, `f`, `o`, and `x` types match the meaning of the corresponding format characters in the historical implementations of *od* except for the default sizes of the fields converted. The `d` format is signed in this volume of IEEE Std 1003.1-200x to match the *printf*( ) notation. (Historical versions of *od* used `d` as a synonym for `u` in this version. The System V implementation uses `s` for signed decimal; BSD uses `i` for signed decimal and `s` for null-terminated strings.) Other than `d` and `u`, all of the type specifiers match format characters in the historical BSD version of **od**.

The sizes of the C-language types **char**, **short**, **int**, **long**, **float**, **double**, and **long double** are used even though it is recognized that there may be zero or more than one compiler for the C language on an implementation and that they may use different sizes for some of these types. (For example, one compiler might use 2 bytes **short**s, 2 bytes **int**s, and 4 bytes **long**s, while another compiler (or an option to the same compiler) uses 2 bytes **short**s, 4 bytes **int**s, and 4 bytes **long**s.) Nonetheless, there has to be a basic size known by the implementation for these types, corresponding to the values reported by invocations of the *getconf* utility when called with *system_var* operands {UCHAR_MAX}, {USHORT_MAX}, {UINT_MAX}, and {ULONG_MAX} for the types **char**, **short**, **int**, and **long**, respectively. There are similar constants required by the ISO C standard, but not required by the System Interfaces volume of IEEE Std 1003.1-200x or this volume of IEEE Std 1003.1-200x. They are {FLT_MANT_DIG}, {DBL_MANT_DIG}, and {LDBL_MANT_DIG} for the types **float**, **double**, and **long double**, respectively. If the optional *c99* utility is provided by the implementation and used as specified by this volume of IEEE Std 1003.1-200x, these are the sizes that would be provided. If an option is used that specifies different sizes for these types, there is no guarantee that the *od* utility is able to interpret binary data output by such a program correctly.

This volume of IEEE Std 1003.1-200x requires that the numeric values of these lengths be recognized by the *od* utility and that symbolic forms also be recognized. Thus, a conforming application can always look at an array of **unsigned long** data elements using *od* –**t** *uL*.

- The method of specifying the format for the address field based on specifying a starting offset in a file unnecessarily tied the two together. The –**A** option now specifies the address base and the –**S** option specifies a starting offset.

- It would be difficult to break the dependence on U.S. ASCII to achieve an internationalized utility. It does not seem to be any harder for *od* to dump characters in the current locale than it is for the *ed* or *sed* **l** commands. The **c** type specifier does this without difficulty and is completely compatible with the historical implementations of the **c** format character when the current locale uses a superset of the ISO/IEC 646:1991 standard as a codeset. The **a** type specifier (from the BSD **a** format character) was left as a portable means to dump ASCII (or more correctly ISO/IEC 646:1991 standard (IRV)) so that headers produced by *pax* could be deciphered even on systems that do not use the ISO/IEC 646:1991 standard as a subset of their base codeset.

The use of `"***"` as an indication of continuation of a multi-byte character in **c** specifier output was chosen based on seeing an implementation that uses this method. The continuation bytes have to be marked in a way that is not ambiguous with another single-byte or multi-byte character.

An early proposal used –**S** and –**n**, respectively, for the –**j** and –**N** options eventually selected. These were changed to avoid conflicts with historical implementations.

The original standard specified **–t o2** as the default when no output type was given. This was changed to **–t oS** (the length of a **short**) to accommodate a supercomputer implementation that historically used 64 bits as its default (and that defined shorts as 64 bits). This change should not affect conforming applications. The requirement to support lengths of 1, 2, and 4 was added at the same time to address an historical implementation that had no two-byte data types in its C compiler.

The use of a basic integer data type is intended to allow the implementation to choose a word size commonly used by applications on that architecture.

FUTURE DIRECTIONS

All option and operand interfaces marked as extensions may be withdrawn in a future issue.

SEE ALSO

c99, *sed*

CHANGE HISTORY

First released in Issue 2.

Issue 5

In the description of the **–c** option, the phrase ‘‘This is equivalent to **–t c**.’’ is deleted.

The FUTURE DIRECTIONS section has been modified.

Issue 6

The *od* utility is changed to remove the assumption that **short** was a two-byte entity, as per the revisions in the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

NAME

paste — merge corresponding or subsequent lines of files

SYNOPSIS

```
paste [–s][–d list] file...
```

DESCRIPTION

The *paste* utility shall concatenate the corresponding lines of the given input files, and writes the resulting lines to standard output.

The default operation of *paste* shall concatenate the corresponding lines of the input files. The <newline> of every line except the line from the last input file shall be replaced with a <tab>.

If an end-of-file condition is detected on one or more input files, but not all input files, *paste* shall behave as though empty lines were read from the files on which end-of-file was detected, unless the **–s** option is specified.

OPTIONS

The *paste* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–d *list* Unless a backslash character appears in *list*, each character in *list* is an element specifying a delimiter character. If a backslash character appears in *list*, the backslash character and one or more characters following it are an element specifying a delimiter character as described below. These elements specify one or more delimiters to use, instead of the default <tab>, to replace the <newline> of the input lines. The elements in *list* shall be used circularly; that is, when the list is exhausted the first element from the list is reused. When the **–s** option is specified:

- The last <newline> in a file shall not be modified.
- The delimiter shall be reset to the first element of list after each *file* operand is processed.

When the **–s** option is not specified:

- The <newline>s in the file specified by the last *file* operand shall not be modified.
- The delimiter shall be reset to the first element of list each time a line is processed from each file.

If a backslash character appears in *list*, it and the character following it shall be used to represent the following delimiter characters:

`\n` <newline>.

`\t` <tab>.

`\\` Backslash character.

`\0` Empty string (not a null character). If `'\0'` is immediately followed by the character `'x'`, the character `'X'`, or any character defined by the *LC_CTYPE* **digit** keyword (see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 7, Locale), the results are unspecified.

If any other characters follow the backslash, the results are unspecified.

–s Concatenate all of the lines of each separate input file in command line order. The <newline> of every line except the last line in each input file shall be replaced with

the <tab>, unless otherwise specified by the **–d** option.

OPERANDS

The following operand shall be supported:

file A pathname of an input file. If '–' is specified for one or more of the *file*s, the standard input shall be used; the standard input shall be read one line at a time, circularly, for each instance of '–'. Implementations shall support pasting of at least 12 *file* operands.

STDIN

The standard input shall be used only if one or more *file* operands is '–'. See the INPUT FILES section.

INPUT FILES

The input files shall be text files, except that line lengths shall be unlimited.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *paste*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Concatenated lines of input files shall be separated by the <tab> (or other characters under the control of the **–d** option) and terminated by a <newline>.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

If one or more input files cannot be opened when the –**s** option is not specified, a diagnostic message shall be written to standard error, but no output is written to standard output. If the –**s** option is specified, the *paste* utility shall provide the default behavior described in Section 1.11 (on page 2221).

APPLICATION USAGE

When the escape sequences of the *list* option-argument are used in a shell script, they must be quoted; otherwise, the shell treats the `'\\'` as a special character.

Conforming applications should only use the specific backslash escaped delimiters presented in this volume of IEEE Std 1003.1-200x. Historical implementations treat `'\x'`, where `'x'` is not in this list, as `'x'`, but future implementations are free to expand this list to recognize other common escapes similar to those accepted by *printf* and other standard utilities.

Most of the standard utilities work on text files. The *cut* utility can be used to turn files with arbitrary line lengths into a set of text files containing the same data. The *paste* utility can be used to create (or recreate) files with arbitrary line lengths. For example, if *file* contains long lines:

```
cut –b 1–500 –n file > file1
cut –b 501– –n file > file2
```

creates **file1** (a text file) with lines no longer than 500 bytes (plus the <newline>) and **file2** that contains the remainder of the data from *file*. Note that **file2** is not a text file if there are lines in *file* that are longer than 500 + {LINE_MAX} bytes. The original file can be recreated from **file1** and **file2** using the command:

```
paste –d "\0" file1 file2 > file
```

The commands:

```
paste –d "\0" ...
paste –d "" ...
```

are not necessarily equivalent; the latter is not specified by this volume of IEEE Std 1003.1-200x and may result in an error. The construct `'\0'` is used to mean ‘‘no separator’’ because historical versions of *paste* did not follow the syntax guidelines, and the command:

```
paste –d"" ...
```

could not be handled properly by *getopt*( ).

EXAMPLES

1. Write out a directory in four columns:

   ```
   ls | paste – – – –
   ```

2. Combine pairs of lines from a file into single lines:

   ```
   paste –s –d "\t\n" file
   ```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cut, *grep*, *pr*

CHANGE HISTORY

First released in Issue 2.

Issue 6

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

NAME

patch — apply changes to files

SYNOPSIS

UP

```
patch [–blNR][ –c| –e| –n][–d dir][–D define][–i patchfile]
    [–o outfile][–p num][–r rejectfile][file]
```

DESCRIPTION

The *patch* utility shall read a source (patch) file containing any of the three forms of difference (diff) listings produced by the *diff* utility (normal, context or in the style of *ed*) 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 application shall ensure that the name of the patch file is determinable for each *diff* listing.)

OPTIONS

The *patch* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–b Save a copy of the original contents of each modified file, before the differences are applied, in a file of the same name with the suffix **.orig** appended to it. If the file already exists, it shall be overwritten; if multiple patches are applied to the same file, the **.orig** file shall be written only for the first patch. When the **–o** *outfile* option is also specified, *file***.orig** shall not be created but, if *outfile* already exists, *outfile***.orig** shall be created.

–c Interpret the patch file as a context difference (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 the EXTENDED DESCRIPTION section.

–D *define* Mark changes with one of the following C preprocessor constructs:

```
#ifdef define
...
#endif
```

```
#ifndef define
...
#endif
```

optionally combined with the C preprocessor construct **#else**.

–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 difference 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 difference.

–N Ignore patches where the differences have already been applied to the file; by default, already-applied patches shall be rejected.

–o *outfile* Instead of modifying the files (specified by the *file* operand or the difference listings) directly, write a copy of the file referenced by each patch, with the appropriate differences applied, to *outfile*. Multiple patches for a single file shall be applied to the intermediate versions of the file created by any previous patches, and shall result in multiple, concatenated versions of the file being written to *outfile*.

–p *num* For all pathnames in the patch file that indicate the names of files to be patched, delete *num* pathname components from the beginning of each pathname. If the pathname in the patch file is absolute, any leading slashes shall be considered the first component (that is, **–p 1** shall remove the leading slashes). Specifying **–p 0** shall cause the full pathname to be used. If **–p** is not specified, only the basename (the final pathname component) shall be used.

–R Reverse the sense of the patch script; that is, assume that the difference script was created from the new version to the old version. The **–R** option cannot be used with *ed* scripts. The *patch* utility shall attempt 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 whether the **–R** option should be set.

–r *rejectfile* Override the default reject filename. In the default case, the reject file shall have the same name as the output file, with the suffix **.rej** appended to it; see **Patch Application** (on page 2893).

OPERANDS

The following operand shall be supported:

file A pathname of a file to patch.

STDIN

See the INPUT FILES section.

INPUT FILES

Input files shall be text files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *patch*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

LC_TIME Determine the locale for recognizing the format of file timestamps written by the *diff* utility in a context-difference input file.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used for diagnostic and informational messages.

OUTPUT FILES

The output of the *patch* utility, the save files (**.orig** suffixes) and the reject files (**.rej** suffixes) shall be text files.

EXTENDED DESCRIPTION

A patchfile may contain patching instructions for more than one file; filenames shall be determined as specified in **Filename Determination** (on page 2893). 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 **Patch Application** (on page 2893). In the absence of a **–r** option, the name of this file shall be formed by appending the suffix **.rej** to the original filename.

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**, 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*.

If all lines (including headers) within a patch begin with the same leading sequence of <blank>s, the *patch* utility shall remove this sequence before proceeding. Within each patch, if the type of difference 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.

Filename Determination

If no *file* operand is specified, *patch* shall perform the following steps to determine the filename to use:

1. If the type of *diff* is context, the *patch* utility shall delete pathname components (as specified by the **–p** option) from the filename on the line beginning with "****", then test for the existence of this file relative to the current directory (or the directory specified with the **–d** option). If the file exists, the *patch* utility shall use this filename.
2. If the type of *diff* is context, the *patch* utility shall delete the pathname components (as specified by the **–p** option) from the filename on the line beginning with "–––", then test for the existence of this file relative to the current directory (or the directory specified with the **–d** option). If the file exists, the *patch* utility shall use this filename.
3. If the header information contains a line beginning with the string **Index:**, the *patch* utility shall delete pathname components (as specified by the **–p** option) from this line, then test for the existence of this file relative to the current directory (or the directory specified with the **–d** option). If the file exists, the *patch* utility shall use this filename.
4. XSI If an **SCCS** directory exists in the current directory, *patch* shall attempt to perform a *get* **–e SCCS/s.***filename* command to retrieve an editable version of the file. If the file exists, the *patch* utility shall use this filename.
5. The *patch* utility shall write a prompt to standard output and request a filename interactively from the controlling terminal (for example, **/dev/tty**).

Patch Application

If the **–c**, **–e**, or **–n** option is present, the *patch* utility shall interpret information within each hunk as a context difference, an *ed* difference or a normal difference, respectively. In the absence of any of these options, the *patch* utility shall determine the type of difference 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, *patch* shall scan both forwards and backwards at least 1 000 bytes for a set of lines that match the hunk context.

If no such place is found and it is a context difference, 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-difference format regardless of the format of the patch file. If the input was a normal or *ed–style* difference, the reject file may contain differences with zero lines of context. The line numbers on the hunks in the reject file may be different from 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.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

26623 1 One or more lines were written to a reject file.

26624 >1 An error occurred.

26625 **CONSEQUENCES OF ERRORS**

26626 Patches that cannot be correctly placed in the file shall be written to a reject file.

26627 **APPLICATION USAGE**

26628 The **–R** option does not work with *ed* scripts because there is too little information to reconstruct
26629 the reverse operation.

26630 The **–p** option makes it possible to customize a patchfile to local user directory structures
26631 without manually editing the patchfile. For example, if the filename in the patch file was:

26632 `/curds/whey/src/blurfl/blurfl.c`

26633 Setting **–p 0** gives the entire pathname unmodified; **–p 1** gives:

26634 `curds/whey/src/blurfl/blurfl.c`

26635 without the leading slash, **–p 4** gives:

26636 `blurfl/blurfl.c`

26637 and not specifying **–p** at all gives:

26638 `blurfl.c .`

26639 **EXAMPLES**

26640 None.

26641 **RATIONALE**

26642 Some of the functionality in historical *patch* implementations was not specified. The following
26643 documents those features present in historical implementations that have not been specified.

26644 A deleted piece of functionality was the `'+'` pseudo-option allowing an additional set of options
26645 and a patch file operand to be given. This was seen as being insufficiently useful to standardize.

26646 In historical implementations, if the string `"Prereq:"` appeared in the header, the *patch* utility
26647 would search for the corresponding version information (the string specified in the header,
26648 delimited by <blank>s or the beginning or end of a line or the file) anywhere in the original file.
26649 This was deleted as too simplistic and insufficiently trustworthy a mechanism to standardize.
26650 For example, if:

26651 `Prereq: 1.2`

26652 were in the header, the presence of a delimited 1.2 anywhere in the file would satisfy the
26653 prerequisite.

26654 The following options were dropped from historical implementations of *patch* as insufficiently
26655 useful to standardize:

26656 **–b** The **–b** option historically provided a method for changing the name extension of
26657 the backup file from the default **.orig**. This option has been modified and retained
26658 in this volume of IEEE Std 1003.1-200x.

26659 **–F** The **–F** option specified the number of lines of a context diff to ignore when
26660 searching for a place to install a patch.

26661 **–f** The **–f** option historically caused *patch* not to request additional information from
26662 the user.

–r The **–r** option historically provided a method of overriding the extension of the reject file from the default **.rej**.

–s The **–s** option historically caused *patch* to work silently unless an error occurred.

–x The **–x** option historically 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 overwrites the new file. The reject file may also exceed this filename limit. It was suggested, due to some historical 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 not to save the files, making *patch* behave more consistently with the other standard utilities.

The **–w** option in early proposals was changed to **–l** to match historical practice.

The **–N** option was included because without it, a non-interactive 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 non-interactively, the **–N** option is required.

Changes to the **–l** option description were proposed to allow matching across <newline>s in addition to just <blank>s. Since this is not historical practice, and since some ambiguities could result, it is suggested that future developments in this area utilize another option letter, such as **–L**.

FUTURE DIRECTIONS

None.

SEE ALSO

ed, *diff*

CHANGE HISTORY

First released in Issue 4.

Issue 5

FUTURE DIRECTIONS section added.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The description of the **–D** option and the steps in **Filename Determination** (on page 2893) are changed to match historical practice as defined in the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

26699 **NAME**

26700 pathchk — check pathnames

26701 **SYNOPSIS**

26702 `pathchk [–p] pathname...`

26703 **DESCRIPTION**

26704 The *pathchk* utility shall check that one or more pathnames are valid (that is, they could be used
26705 to access or create a file without causing syntax errors) and portable (that is, no filename
26706 truncation results). More extensive portability checks are provided by the **–p** option.

26707 By default, the *pathchk* utility shall check each component of each *pathname* operand based on the
26708 underlying file system. A diagnostic shall be written for each *pathname* operand that:

26709 • Is longer than {PATH_MAX} bytes (see **Pathname Variable Values** in the Base Definitions
26710 volume of IEEE Std 1003.1-200x, Chapter 13, Headers, **<limits.h>**)

26711 • Contains any component longer than {NAME_MAX} bytes in its containing directory

26712 • Contains any component in a directory that is not searchable

26713 • Contains any character in any component that is not valid in its containing directory

26714 The format of the diagnostic message is not specified, but shall indicate the error detected and
26715 the corresponding *pathname* operand.

26716 It shall not be considered an error if one or more components of a *pathname* operand do not exist
26717 as long as a file matching the pathname specified by the missing components could be created
26718 that does not violate any of the checks specified above.

26719 **OPTIONS**

26720 The *pathchk* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
26721 12.2, Utility Syntax Guidelines.

26722 The following option shall be supported:

26723 **–p** Instead of performing checks based on the underlying file system, write a
26724 diagnostic for each *pathname* operand that:

26725 • Is longer than {_POSIX_PATH_MAX} bytes (see **Minimum Values** in the Base
26726 Definitions volume of IEEE Std 1003.1-200x, Chapter 13, Headers, **<limits.h>**)

26727 • Contains any component longer than {_POSIX_NAME_MAX} bytes

26728 • Contains any character in any component that is not in the portable filename
26729 character set

26730 **OPERANDS**

26731 The following operand shall be supported:

26732 *pathname* A pathname to be checked.

26733 **STDIN**

26734 Not used.

26735 **INPUT FILES**

26736 None.

26737 **ENVIRONMENT VARIABLES**

26738 The following environment variables shall affect the execution of *pathchk*:

26739 *LANG* Provide a default value for the internationalization variables that are unset or null.
26740 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,

Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All *pathname* operands passed all of the checks.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *test* utility can be used to determine whether a given pathname names an existing file; it does not, however, give any indication of whether or not any component of the pathname was truncated in a directory where the _POSIX_NO_TRUNC feature is not in effect. The *pathchk* utility does not check for file existence; it performs checks to determine whether a pathname does exist or could be created with no pathname component truncation.

The *noclobber* option in the shell (see the *set* (on page 2287) special built-in) can be used to atomically create a file. As with all file creation semantics in the System Interfaces volume of IEEE Std 1003.1-200x, it guarantees atomic creation, but still depends on applications to agree on conventions and cooperate on the use of files after they have been created.

EXAMPLES

To verify that all pathnames in an imported data interchange archive are legitimate and unambiguous on the current system:

```
pax –f archive | sed –e '/ == .*/s///' | xargs pathchk
if [ $? –eq 0 ]
then
    pax –r –f archive
```

```
else
    echo Investigate problems before importing files.
    exit 1
fi
```

To verify that all files in the current directory hierarchy could be moved to any system conforming to the System Interfaces volume of IEEE Std 1003.1-200x that also supports the *pax* utility:

```
find . –print | xargs pathchk –p
if [ $? –eq 0 ]
then
    pax –w –f archive .
else
    echo Portable archive cannot be created.
    exit 1
fi
```

To verify that a user-supplied pathname names a readable file and that the application can create a file extending the given path without truncation and without overwriting any existing file:

```
case $– in
    *C*)    reset="";;
    *)      reset="set +C"
            set –C;;
esac
test –r "$path" && pathchk "$path.out" &&
    rm "$path.out" > "$path.out"
if [ $? –ne 0 ]; then
    printf "%s: %s not found or %s.out fails \
creation checks.\n" $0 "$path" "$path"
    $reset    # Reset the noclobber option in case a trap
              # on EXIT depends on it.
    exit 1
fi
$reset
PROCESSING < "$path" > "$path.out"
```

The following assumptions are made in this example:

1. **PROCESSING** represents the code that is used by the application to use **$path** once it is verified that **$path.out** works as intended.
2. The state of the *noclobber* option is unknown when this code is invoked and should be set on exit to the state it was in when this code was invoked. (The **reset** variable is used in this example to restore the initial state.)
3. Note the usage of:

   ```
   rm "$path.out" > "$path.out"
   ```

   a. The *pathchk* command has already verified, at this point, that **$path.out** is not truncated.

   b. With the *noclobber* option set, the shell verifies that **$path.out** does not already exist before invoking *rm*.

c. If the shell succeeded in creating **$path.out**, *rm* removes it so that the application can create the file again in the **PROCESSING** step.

d. If the **PROCESSING** step wants the file to exist already when it is invoked, the:

```
rm "$path.out" > "$path.out"
```

should be replaced with:

```
> "$path.out"
```

which verifies that the file did not already exist, but leaves **$path.out** in place for use by **PROCESSING**.

RATIONALE

The *pathchk* utility is new, commissioned for this volume of IEEE Std 1003.1-200x. It, along with the *set* **–C**(*noclobber*) option added to the shell, replaces the *mktemp*, *validfnam*, and *create* utilities that appeared in early proposals. All of these utilities were attempts to solve several common problems:

- Verify the validity (for several different definitions of ‘‘valid’’) of a pathname supplied by a user, generated by an application, or imported from an external source.
- Atomically create a file.
- Perform various string handling functions to generate a temporary filename.

The *create* utility, included in an early proposal, provided checking and atomic creation in a single invocation of the utility; these are orthogonal issues and need not be grouped into a single utility. Note that the *noclobber* option also provides a way of creating a lock for process synchronization; since it provides an atomic *create*, there is no race between a test for existence and the following creation if it did not exist.

Having a function like *tmpnam*( ) in the ISO C standard is important in many high-level languages. The shell programming language, however, has built-in string manipulation facilities, making it very easy to construct temporary filenames. The names needed obviously depend on the application, but are frequently of a form similar to:

$TMPDIR**/*application_abbreviation***$$.*suffix*

In cases where there is likely to be contention for a given suffix, a simple shell *for* or *while* loop can be used with the shell *noclobber* option to create a file without risk of collisions, as long as applications trying to use the same filename name space are cooperating on the use of files after they have been created.

FUTURE DIRECTIONS

None.

SEE ALSO

test, Section 2.7 (on page 2244)

CHANGE HISTORY

First released in Issue 4.

NAME

pax — portable archive interchange

SYNOPSIS

```
pax [–cdnv][–H|–L][–f archive][–s replstr]...[pattern...]

pax –r[–cdiknuv][–H|–L][–f archive][–o options]...[–p string]...
    [–s replstr]...[pattern...]

pax –w[–dituvX][–H|–L][–b blocksize][[–a][–f archive][–o options]...
    [–s replstr]...[–x format][file...]

pax –r –w[–diklntuvX][–H|–L][–p string]...[–s replstr]...
    [file...] directory
```

DESCRIPTION

The *pax* utility shall read, write, and write lists of the members of archive files and copy directory hierarchies. A variety of archive formats shall be supported; see the **–x** *format* option.

The action to be taken depends on the presence of the **–r** and **–w** options. The four combinations of **–r** and **–w** are referred to as the four modes of operation: **list**, **read**, **write**, and **copy** modes, corresponding respectively to the four forms shown in the SYNOPSIS section.

list In **list** mode (when neither **–r** nor **–w** are specified), *pax* shall write the names of the members of the archive file read from the standard input, with pathnames matching the specified patterns, to standard output. If a named file is of type directory, the file hierarchy rooted at that file shall be listed as well.

read In **read** mode (when **–r** is specified, but **–w** is not), *pax* shall extract the members of the archive file read from the standard input, with pathnames matching the specified patterns. If an extracted file is of type directory, the file hierarchy rooted at that file shall be extracted as well. The extracted files shall be created performing pathname resolution with the directory in which *pax* was invoked as the current working directory.

If an attempt is made to extract a directory when the directory already exists, this shall not be considered to be an error. If an attempt is made to extract a FIFO when the FIFO already exists, this shall not be considered to be an error.

The ownership, access, and modification times, and file mode of the restored files are discussed under the **–p** option.

write In **write** mode (when **–w** is specified, but **–r** is not), *pax* shall write the contents of the *file* operands to the standard output in an archive format. If no *file* operands are specified, a list of files to copy, one per line, shall be read from the standard input. A file of type directory shall include all of the files in the file hierarchy rooted at the file.

copy In **copy** mode (when both **–r** and **–w** are specified), *pax* shall copy the *file* operands to the destination directory.

If no *file* operands are specified, a list of files to copy, one per line, shall be read from the standard input. A file of type directory shall include all of the files in the file hierarchy rooted at the file.

The effect of the **copy** shall be as if the copied files were written to an archive file and then subsequently extracted, except that there may be hard links between the original and the copied files. If the destination directory is a subdirectory of one of the files to be copied, the results are unspecified. If the destination directory is a

file of a type not defined by the System Interfaces volume of IEEE Std 1003.1-200x, the results are implementation-defined; otherwise, it shall be an error for the file named by the *directory* operand not to exist, not be writable by the user, or not be a file of type directory.

In **read** or **copy** modes, if intermediate directories are necessary to extract an archive member, *pax* shall perform actions equivalent to the *mkdir*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x, called with the following arguments:

- The intermediate directory used as the *path* argument
- The value of the bitwise-inclusive OR of S_IRWXU, S_IRWXG, and S_IRWXO as the *mode* argument

If any specified *pattern* or *file* operands are not matched by at least one file or archive member, *pax* shall write a diagnostic message to standard error for each one that did not match and exit with a non-zero exit status.

The archive formats described in the EXTENDED DESCRIPTION section shall be automatically detected on input. The default output archive format shall be implementation-defined.

A single archive can span multiple files. The *pax* utility shall determine, in an implementation-defined manner, what file to read or write as the next file.

If the selected archive format supports the specification of linked files, it shall be an error if these files cannot be linked when the archive is extracted. For archive formats that do not store file contents with each name that causes a hard link, if the file that contains the data is not extracted during this *pax* session, either the data shall be restored from the original file, or a diagnostic message shall be displayed with the name of a file that can be used to extract the data. In traversing directories, *pax* shall detect infinite loops; that is, entering a previously visited directory that is an ancestor of the last file visited. When it detects an infinite loop, *pax* shall write a diagnostic message to standard error and shall terminate.

OPTIONS

The *pax* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines, except that the order of presentation of the **–o**, **–p**, and **–s** options is significant.

The following options shall be supported:

–r Read an archive file from standard input.

–w Write files to the standard output in the specified archive format.

–a Append files to the end of the archive. It is implementation-defined which devices on the system support appending. Additional file formats unspecified by this volume of IEEE Std 1003.1-200x may impose restrictions on appending.

–b *blocksize* Block the output at a positive decimal integer number of bytes per write to the archive file. Devices and archive formats may impose restrictions on blocking. Blocking shall be automatically determined on input. Conforming applications shall not specify a *blocksize* value larger than 32 256. Default blocking when creating archives depends on the archive format. (See the **–x** option below.)

–c Match all file or archive members except those specified by the *pattern* or *file* operands.

–d Cause files of type directory being copied or archived or archive members of type directory being extracted or listed to match only the file or archive member itself and not the file hierarchy rooted at the file.

–f *archive* Specify the pathname of the input or output archive, overriding the default standard input (in **list** or **read** modes) or standard output (**write** mode).

–H If a symbolic link referencing a file of type directory is specified on the command line, *pax* shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. Otherwise, if a symbolic link referencing a file of any other file type which *pax* can normally archive is specified on the command line, then *pax* shall archive the file referenced by the link, using the name of the link. The default behavior shall be to archive the symbolic link itself.

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

The results of extracting a hard link to a file that has been renamed during extraction are unspecified.

–k Prevent the overwriting of existing files.

–l (The letter ell.) In **copy** mode, hard links shall be made between the source and destination file hierarchies whenever possible. If specified in conjunction with **–H** or **–L**, when a symbolic link is encountered, the hard link created in the destination file hierarchy shall be to the file referenced by the symbolic link. If specified when neither **–H** nor **–L** is specified, when a symbolic link is encountered, the implementation shall create a hard link to the symbolic link in the source file hierarchy or copy the symbolic link to the destination.

–L If a symbolic link referencing a file of type directory is specified on the command line or encountered during the traversal of a file hierarchy, *pax* shall archive the file hierarchy rooted in the file referenced by the link, using the name of the link as the root of the file hierarchy. Otherwise, if a symbolic link referencing a file of any other file type which *pax* can normally archive is specified on the command line or encountered during the traversal of a file hierarchy, *pax* shall archive the file referenced by the link, using the name of the link. The default behavior shall be to archive the symbolic link itself.

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

–o *options* Provide information to the implementation to modify the algorithm for extracting or writing files. The value of *options* shall consist of one or more comma-separated keywords of the form:

keyword`[[:]=`*value*`][,`*keyword*`[[:]=`*value*`], ...]`

Some keywords apply only to certain file formats, as indicated with each description. Use of keywords that are inapplicable to the file format being processed produces undefined results.

Keywords in the *options* argument shall be a string that would be a valid portable filename as described in the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.276, Portable Filename Character Set.

Note: Keywords are not expected to be filenames, merely to follow the same character composition rules as portable filenames.

Keywords can be preceded with white space. The *value* field shall consist of zero or more characters; within *value*, the application shall precede any literal comma with a backslash, which shall be ignored, but preserves the comma as part of *value*. A comma as the final character, or a comma followed solely by white space as the final characters, in *options* shall be ignored. Multiple **–o** options can be specified; if keywords given to these multiple **–o** options conflict, the keywords and values appearing later in command line sequence shall take precedence and the earlier shall be silently ignored. The following keyword values of *options* shall be supported for the file formats as indicated:

delete=*pattern*
(Applicable only to the **–x** *pax* format.) When used in **write** or **copy** mode, *pax* shall omit from extended header records that it produces any keywords matching the string pattern. When used in **read** or **list** mode, *pax* shall ignore any keywords matching the string pattern in the extended header records. In both cases, matching shall be performed using the pattern matching notation described in Section 2.13.1 (on page 2264) and Section 2.13.2 (on page 2264). For example:

```
–o delete=security.*
```

would suppress security-related information. See **pax Extended Header** (on page 2913) for extended header record keyword usage.

exthdr.name=*string*
(Applicable only to the **–x** *pax* format.) This keyword allows user control over the name that is written into the **ustar** header blocks for the extended header produced under the circumstances described in **pax Header Block** (on page 2912). The name shall be the contents of *string*, after the following character substitutions have been made:

string **Includes:**	**Replaced By:**
%d	The directory name of the file, equivalent to the result of the *dirname* utility on the translated pathname.
%f	The filename of the file, equivalent to the result of the *basename* utility on the translated pathname.
%%	A '%' character.

Any other '%' characters in *string* produce undefined results.

If no **–o exthdr.name=***string* is specified, *pax* shall use the following default value:

```
%d/PaxHeaders/%f
```

globexthdr.name=*string*
(Applicable only to the **–x** *pax* format.) When used in **write** or **copy** mode with the appropriate options, *pax* shall create global extended header records with **ustar** header blocks that will be treated as regular files by previous versions of

27050 *pax*. This keyword allows user control over the name that is written into the
27051 **ustar** header blocks for global extended header records. The name shall be the
27052 contents of string, after the following character substitutions have been made:

27053 27054 *string* **Includes:**	**Replaced By:**
27055 %n	27056 An integer that represents the sequence number of the global extended header record in the archive, starting at 1.
27057 %%	A '%' character.

27058 Any other '%' characters in *string* produce undefined results.

27059 If no –**o globexthdr.name=***string* is specified, *pax* shall use the following
27060 default value:

27061 `$TMPDIR/GlobalHead.%n`

27062 where $*TMPDIR* represents the value of the *TMPDIR* environment variable. If
27063 *TMPDIR* is not set, *pax* shall use **/tmp**.

27064 **invalid**=*action*
27065 (Applicable only to the –**x** *pax* format.) This keyword allows user control over
27066 the action *pax* takes upon encountering values in an extended header record
27067 that, in **read** or **copy** mode, are invalid in the destination hierarchy or, in **list**
27068 mode, cannot be written in the codeset and current locale of the
27069 implementation. The following are invalid values that shall be recognized by
27070 *pax*:

27071 — In **read** or **copy** mode, a filename or link name that contains character
27072 encodings invalid in the destination hierarchy. (For example, the name
27073 may contain embedded NULs.)

27074 — In **read** or **copy** mode, a filename or link name that is longer than the
27075 maximum allowed in the destination hierarchy (for either a pathname
27076 component or the entire pathname).

27077 — In **list** mode, any character string value (filename, link name, user name,
27078 and so on) that cannot be written in the codeset and current locale of the
27079 implementation.

27080 The following mutually-exclusive values of the *action* argument are
27081 supported:

27082 **bypass** In **read** or **copy** mode, *pax* shall bypass the file, causing no
27083 change to the destination hierarchy. In **list** mode, *pax* shall write
27084 all requested valid values for the file, but its method for writing
27085 invalid values is unspecified.

27086 **rename** In **read** or **copy** mode, *pax* shall act as if the –**i** option were in
27087 effect for each file with invalid filename or link name values,
27088 allowing the user to provide a replacement name interactively.
27089 In **list** mode, *pax* shall behave identically to the **bypass** action.

27090 **UTF-8** When used in **read**, **copy**, or **list** mode and a filename, link
27091 name, owner name, or any other field in an extended header
27092 record cannot be translated from the *pax* UTF-8 codeset format
27093 to the codeset and current locale of the implementation, *pax*
27094 shall use the actual UTF-8 encoding for the name.

write In **read** or **copy** mode, *pax* shall write the file, translating or truncating the name, regardless of whether this may overwrite an existing file with a valid name. In **list** mode, *pax* shall behave identically to the **bypass** action.

If no –**o invalid=** option is specified, *pax* shall act as if –**oinvalid=bypass** were specified. Any overwriting of existing files that may be allowed by the –**oinvalid=** actions shall be subject to permission (–**p**) and modification time (–**u**) restrictions, and shall be suppressed if the –**k** option is also specified.

linkdata
(Applicable only to the –**x** *pax* format.) In **write** mode, *pax* shall write the contents of a file to the archive even when that file is merely a hard link to a file whose contents have already been written to the archive.

listopt=*format*
This keyword specifies the output format of the table of contents produced when the –**v** option is specified in **list** mode. See **List Mode Format Specifications** (on page 2908). To avoid ambiguity, the **listopt=***format* shall be the only or final **keyword=***value* pair in a –**o** option-argument; all characters in the remainder of the option-argument shall be considered part of the format string. When multiple –**olistopt=***format* options are specified, the format strings shall be considered a single, concatenated string, evaluated in command line order.

times
(Applicable only to the –**x** *pax* format.) When used in **write** or **copy** mode, *pax* shall include **atime**, **ctime**, and **mtime** extended header records for each file. See **pax Extended Header File Times** (on page 2916).

In addition to these keywords, if the –**x** *pax* format is specified, any of the keywords and values defined in **pax Extended Header** (on page 2913), including implementation extensions, can be used in –**o** option-arguments, in either of two modes:

keyword=*value*
When used in **write** or **copy** mode, these keyword/value pairs shall be included at the beginning of the archive as **typeflag g** global extended header records. When used in **read** or **list** mode, these keyword/value pairs shall act as if they had been at the beginning of the archive as **typeflag g** global extended header records.

keyword:=*value*
When used in **write** or **copy** mode, these keyword/value pairs shall be included as records at the beginning of a **typeflag x** extended header for each file. (This shall be equivalent to the equal-sign form except that it creates no **typeflag g** global extended header records.) When used in **read** or **list** mode, these keyword/value pairs shall act as if they were included as records at the end of each extended header; thus, they shall override any global or file-specific extended header record keywords of the same names. For example, in the command:

```
pax –r –o "
gname:=mygroup,
" <archive
```

the group name will be forced to a new value for all files read from the archive.

The precedences of –**o** keywords over various fields in the archive are described in **pax Extended Header Keyword Precedence** (on page 2916).

–**p** *string* Specify one or more file characteristic options (privileges). The *string* option-argument shall be a string specifying file characteristics to be retained or discarded on extraction. The string shall consist of the specification characters `a`, `e`, `m`, `o`, and `p`. Other implementation-defined characters can be included. Multiple characteristics can be concatenated within the same string and multiple –**p** options can be specified. The meaning of the specification characters are as follows:

`a` Do not preserve file access times.

`e` Preserve the user ID, group ID, file mode bits (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.168, File Mode Bits), access time, modification time, and any other implementation-defined file characteristics.

`m` Do not preserve file modification times.

`o` Preserve the user ID and group ID.

`p` Preserve the file mode bits. Other implementation-defined file mode attributes may be preserved.

In the preceding list, ‘‘preserve’’ indicates that an attribute stored in the archive shall be given to the extracted file, subject to the permissions of the invoking process. The access and modification times of the file shall be preserved unless otherwise specified with the –**p** option or not stored in the archive. All attributes that are not preserved shall be determined as part of the normal file creation action (see Section 1.7.1.4 (on page 2204)).

If neither the **e** nor the **o** specification character is specified, or the user ID and group ID are not preserved for any reason, *pax* shall not set the S_ISUID and S_ISGID bits of the file mode.

If the preservation of any of these items fails for any reason, *pax* shall write a diagnostic message to standard error. Failure to preserve these items shall affect the final exit status, but shall not cause the extracted file to be deleted.

If file characteristic letters in any of the *string* option-arguments are duplicated or conflict with each other, the ones given last shall take precedence. For example, if –**p** `eme` is specified, file modification times are preserved.

–**s** *replstr* Modify file or archive member names named by *pattern* or *file* operands according to the substitution expression *replstr*, using the syntax of the *ed* utility. The concepts of ‘‘address’’ and ‘‘line’’ are meaningless in the context of the *pax* utility, and shall not be supplied. The format shall be:

–s */old/new/*[gp]

where as in *ed*, *old* is a basic regular expression and *new* can contain an ampersand, `'\n'` (where *n* is a digit) backreferences, or subexpression matching. The *old* string also shall be permitted to contain <newline>s.

Any non-null character can be used as a delimiter (`'/'` shown here). Multiple –**s** expressions can be specified; the expressions shall be applied in the order specified, terminating with the first successful substitution. The optional trailing `'g'` is as defined in the *ed* utility. The optional trailing `'p'` shall cause successful

substitutions to be written to standard error. File or archive member names that substitute to the empty string shall be ignored when reading and writing archives.

–t When reading files from the file system, and if the user has the permissions required by *utime*( ) to do so, set the access time of each file read to the access time that it had before being read by *pax*.

–u Ignore files that are older (having a less recent file modification time) than a pre-existing file or archive member with the same name. In **read** mode, an archive member with the same name as a file in the file system shall be extracted if the archive member is newer than the file. In **write** mode, an archive file member with the same name as a file in the file system shall be superseded if the file is newer than the archive member. If **–a** is also specified, this is accomplished by appending to the archive; otherwise, it is unspecified whether this is accomplished by actual replacement in the archive or by appending to the archive. In **copy** mode, the file in the destination hierarchy shall be replaced by the file in the source hierarchy or by a link to the file in the source hierarchy if the file in the source hierarchy is newer.

–v In **list** mode, produce a verbose table of contents (see the STDOUT section). Otherwise, write archive member pathnames to standard error (see the STDERR section).

–x *format* Specify the output archive format. The *pax* utility shall support the following formats:

cpio The *cpio* interchange format; see the EXTENDED DESCRIPTION section. The default *blocksize* for this format for character special archive files shall be 5 120. Implementations shall support all *blocksize* values less than or equal to 32 256 that are multiples of 512.

pax The *pax* interchange format; see the EXTENDED DESCRIPTION section. The default *blocksize* for this format for character special archive files shall be 5 120. Implementations shall support all *blocksize* values less than or equal to 32 256 that are multiples of 512.

ustar The *tar* interchange format; see the EXTENDED DESCRIPTION section. The default *blocksize* for this format for character special archive files shall be 10 240. Implementations shall support all *blocksize* values less than or equal to 32 256 that are multiples of 512.

Implementation-defined formats shall specify a default block size as well as any other block sizes supported for character special archive files.

Any attempt to append to an archive file in a format different from the existing archive format shall cause *pax* to exit immediately with a non-zero exit status.

In **copy** mode, if no **–x** format is specified, *pax* shall behave as if **–x***pax* were specified.

–X When traversing the file hierarchy specified by a pathname, *pax* shall not descend into directories that have a different device ID (*st_dev*; see the System Interfaces volume of IEEE Std 1003.1-200x, *stat*( )).

The options that operate on the names of files or archive members (**–c**, **–i**, **–n**, **–s**, **–u**, and **–v**) shall interact as follows. In **read** mode, the archive members shall be selected based on the user-specified *pattern* operands as modified by the **–c**, **–n**, and **–u** options. Then, any **–s** and **–i** options shall modify, in that order, the names of the selected files. The **–v** option shall write names resulting from these modifications.

In **write** mode, the files shall be selected based on the user-specified pathnames as modified by the –**n** and –**u** options. Then, any –**s** and –**i** options shall modify, in that order, the names of these selected files. The –**v** option shall write names resulting from these modifications.

If both the –**u** and –**n** options are specified, *pax* shall not consider a file selected unless it is newer than the file to which it is compared.

List Mode Format Specifications

In **list** mode with the –**o listopt=***format* option, the *format* argument shall be applied for each selected file. The *pax* utility shall append a <newline> to the **listopt** output for each selected file. The format argument shall be used as the *format* string described in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation, with the exceptions 1. through 5. defined in the EXTENDED DESCRIPTION section of *printf*, plus the following exceptions:

6. The sequence (*keyword*) can occur before a format conversion specifier. The conversion argument is defined by the value of *keyword*. The implementation shall support the following keywords:

   — Any of the Field Name entries in Table 4-13 (on page 2917) and Table 4-15 (on page 2920). The implementation may support the *cpio* keywords without the leading **c_** in addition to the form required by Table 4-16 (on page 2921).

   — Any keyword defined for the extended header in **pax Extended Header** (on page 2913).

   — Any keyword provided as an implementation-defined extension within the extended header defined in **pax Extended Header** (on page 2913).

   For example, the sequence `"%(charset)s"` is the string value of the name of the character set in the extended header.

   The result of the keyword conversion argument shall be the value from the applicable header field or extended header, without any trailing NULs.

   All keyword values used as conversion arguments shall be translated from the UTF-8 encoding to the character set appropriate for the local file system, user database, and so on, as applicable.

7. An additional conversion specifier character, `T`, shall be used to specify time formats. The `T` conversion specifier character can be preceded by the sequence (*keyword=subformat*), where *subformat* is a date format as defined by *date* operands. The default *keyword* shall be **mtime** and the default subformat shall be:

   ```
   %b %e %H:%M %Y
   ```

8. An additional conversion specifier character, `M`, shall be used to specify the file mode string as defined in *ls* Standard Output. If (*keyword*) is omitted, the **mode** keyword shall be used. For example, `%.1M` writes the single character corresponding to the <*entry type*> field of the *ls* –**l** command.

9. An additional conversion specifier character, `D`, shall be used to specify the device for block or special files, if applicable, in an implementation-defined format. If not applicable, and (*keyword*) is specified, then this conversion shall be equivalent to `%(`*keyword*`)u`. If not applicable, and (*keyword*) is omitted, then this conversion shall be equivalent to <space>.

10. An additional conversion specifier character, `F`, shall be used to specify a pathname. The `F` conversion character can be preceded by a sequence of comma-separated keywords:

    ```
    (keyword[,keyword] ... )
    ```

The values for all the keywords that are non-null shall be concatenated together, each separated by a `'//'`. The default shall be (**path**) if the keyword **path** is defined; otherwise, the default shall be (**prefix**,**name**).

11. An additional conversion specifier character, `L`, shall be used to specify a symbolic line expansion. If the current file is a symbolic link, then `%L` shall expand to:

    `"%s –> %s", <value of keyword>, <contents of link>`

    Otherwise, the `%L` conversion specification shall be the equivalent of `%F`.

OPERANDS

The following operands shall be supported:

directory The destination directory pathname for **copy** mode.

file A pathname of a file to be copied or archived.

pattern A pattern matching one or more pathnames of archive members. A pattern must be given in the name-generating notation of the pattern matching notation in Section 2.13 (on page 2264), including the filename expansion rules in Section 2.13.3 (on page 2265). The default, if no *pattern* is specified, is to select all members in the archive.

STDIN

In **write** mode, the standard input shall be used only if no *file* operands are specified. It shall be a text file containing a list of pathnames, one per line, without leading or trailing <blank>s.

In **list** and **read** modes, if **–f** is not specified, the standard input shall be an archive file.

Otherwise, the standard input shall not be used.

INPUT FILES

The input file named by the *archive* option-argument, or standard input when the archive is read from there, shall be a file formatted according to one of the specifications in the EXTENDED DESCRIPTION section or some other implementation-defined format.

The file **/dev/tty** shall be used to write prompts and read responses.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *pax*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes and multi-character collating elements used in the pattern matching expressions for the *pattern* operand, the basic regular expression for the **–s** option, and the extended regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES* category.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), the behavior of character classes used in the extended regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES*

category, and pattern matching.

LC_MESSAGES
Determine the locale for the processing of affirmative responses that should be used to affect the format and contents of diagnostic messages written to standard error.

LC_TIME Determine the format and contents of date and time strings when the **–v** option is specified.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

TMPDIR Determine the pathname that provides part of the default global extended header record file, as described for the **–o globexthdr=** keyword as described in the OPTIONS section.

TZ Determine the timezone used to calculate date and time strings when the **–v** option is specified. If *TZ* is unset or null, an unspecified default timezone shall be used.

ASYNCHRONOUS EVENTS

Default.

STDOUT

In **write** mode, if **–f** is not specified, the standard output shall be the archive formatted according to one of the specifications in the EXTENDED DESCRIPTION section, or some other implementation-defined format (see **–x** *format*).

In **list** mode, when the **–olistopt**=*format* has been specified, the selected archive members shall be written to standard output using the format described under **List Mode Format Specifications** (on page 2908). In **list** mode without the **–olistopt**=*format* option, the table of contents of the selected archive members shall be written to standard output using the following format:

```
"%s\n", <path name>
```

If the **–v** option is specified in **list** mode, the table of contents of the selected archive members shall be written to standard output using the following formats.

For pathnames representing hard links to previous members of the archive:

```
"%sΔ==Δ%s\n", <ls –l listing>, <linkname>
```

For all other pathnames:

```
"%s\n", <ls –l listing>
```

where <*ls* –l *listing*> shall be the format specified by the *ls* utility with the **–l** option. When writing pathnames in this format, it is unspecified what is written for fields for which the underlying archive format does not have the correct information, although the correct number of <blank>-separated fields shall be written.

In **list** mode, standard output shall not be buffered more than a line at a time.

STDERR

If **–v** is specified in **read**, **write**, or **copy** modes, *pax* shall write the pathnames it processes to the standard error output using the following format:

```
"%s\n", <pathname>
```

These pathnames shall be written as soon as processing is begun on the file or archive member, and shall be flushed to standard error. The trailing <newline>, which shall not be buffered, is written when the file has been read or written.

If the **–s** option is specified, and the replacement string has a trailing `'p'`, substitutions shall be written to standard error in the following format:

`"%s`Δ`>>`Δ`%s\n", <original pathname>, <new pathname>`

In all operating modes of *pax*, optional messages of unspecified format concerning the input archive format and volume number, the number of files, blocks, volumes, and media parts as well as other diagnostic messages may be written to standard error.

In all formats, for both standard output and standard error, it is unspecified how non-printable characters in pathnames or link names are written.

When *pax* is in **read** mode or **list** mode, using the **–xpax** archive format, and a filename, link name, owner name, or any other field in an extended header record cannot be translated from the *pax* UTF-8 codeset format to the codeset and current locale of the implementation, *pax* shall write a diagnostic message to standard error, shall process the file as described for the **–o invalid=**option, and then shall process the next file in the archive.

OUTPUT FILES

In **read** mode, the extracted output files shall be of the archived file type. In **copy** mode, the copied output files shall be the type of the file being copied. In either mode, existing files in the destination hierarchy shall be overwritten only when all permission (**–p**), modification time (**–u**), and invalid-value (**–oinvalid**=) tests allow it.

In **write** mode, the output file named by the **–f** option-argument shall be a file formatted according to one of the specifications in the EXTENDED DESCRIPTION section, or some other implementation-defined format.

EXTENDED DESCRIPTION

pax Interchange Format

A *pax* archive tape or file produced in the **–x***pax* format shall contain a series of blocks. The physical layout of the archive shall be identical to the **ustar** format described in **ustar Interchange Format** (on page 2916). Each file archived shall be represented by the following sequence:

- An optional header block with extended header records. This header block is of the form described in **pax Header Block** (on page 2912), with a *typeflag* value of **x** or **g**. The extended header records, described in **pax Extended Header** (on page 2913), shall be included as the data for this header block.
- A header block that describes the file. Any fields in the preceding optional extended header shall override the associated fields in this header block for this file.
- Zero or more blocks that contain the contents of the file.

At the end of the archive file there shall be two 512-byte blocks filled with binary zeroes, interpreted as an end-of-archive indicator.

A schematic of an example archive with global extended header records and two actual files is shown in Figure 4-1 (on page 2912). In the example, the second file in the archive has no extended header preceding it, presumably because it has no need for extended attributes.

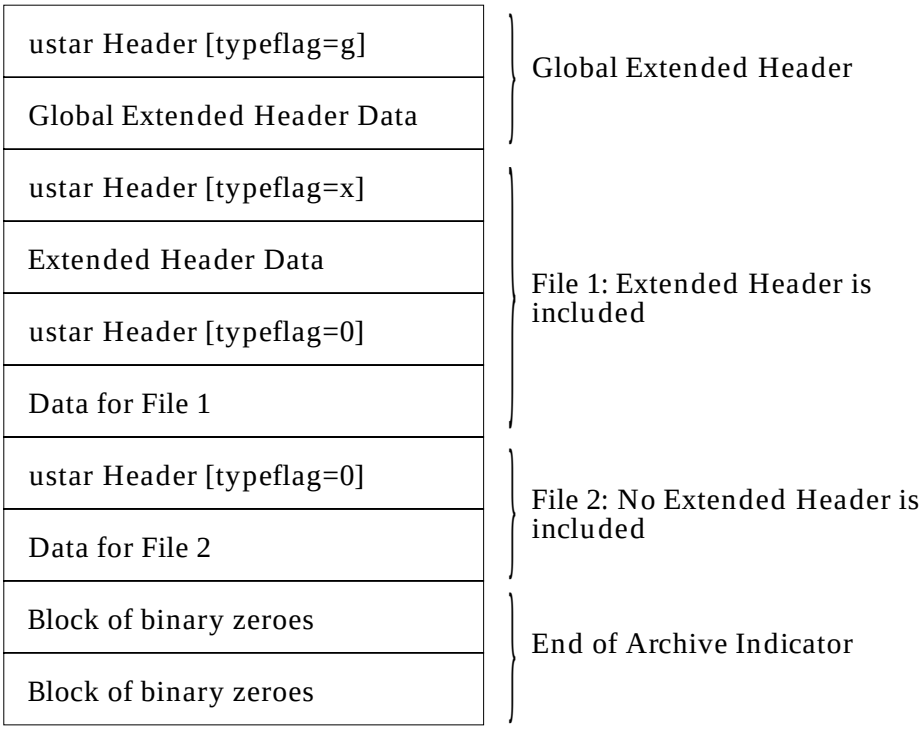

Figure 4-1 pax Format Archive Example

pax Header Block

The *pax* header block shall be identical to the **ustar** header block described in **ustar Interchange Format** (on page 2916), except that two additional *typeflag* values are defined:

x Represents extended header records for the following file in the archive (which shall have its own **ustar** header block). The format of these extended header records shall be as described in **pax Extended Header** (on page 2913).

g Represents global extended header records for the following files in the archive. The format of these extended header records shall be as described in **pax Extended Header** (on page 2913). Each value shall affect all subsequent files that do not override that value in their own extended header record and until another global extended header record is reached that provides another value for the same field. The *typeflag* **g** global headers should not be used with interchange media that could suffer partial data loss in transporting the archive.

For both of these types, the *size* field shall be the size of the extended header records in octets. The other fields in the header block are not meaningful to this version of the *pax* utility. However, if this archive is read by a *pax* utility conforming to a previous version of IEEE Std 1003.1-200x, the header block fields are used to create a regular file that contains the extended header records as data. Therefore, header block field values should be selected to provide reasonable file access to this regular file.

A further difference from the **ustar** header block is that data blocks for files of *typeflag* 1 (the digit one) (hard link) may be included, which means that the size field may be greater than zero. Archives created by **pax –o linkdata** shall include these data blocks with the hard links.

27425 **pax Extended Header**

27426 A *pax* extended header contains values that are inappropriate for the **ustar** header block because
27427 of limitations in that format: fields requiring a character encoding other than that described in
27428 the ISO/IEC 646:1991 standard, fields representing file attributes not described in the **ustar**
27429 header, and fields whose format or length do not fit the requirements of the **ustar** header. The
27430 values in an extended header add attributes to the following file (or files; see the description of
27431 the *typeflag* **g** header block) or override values in the following header block(s), as indicated in
27432 the following list of keywords.

27433 An extended header shall consist of one or more records, each constructed as follows:

27434 `"%d %s=%s\n", <length>, <keyword>, <value>`

27435 The extended header records shall be encoded according to the ISO/IEC 10646-1:2000 standard
27436 (UTF-8). The <*length*> field, <blank>, equals sign, and <newline> shown shall be limited to the
27437 portable character set, as encoded in UTF-8. The <*keyword*> and <*value*> fields can be any UTF-8
27438 characters. The <*length*> field shall be the decimal length of the extended header record in octets,
27439 including the trailing <newline>.

27440 The <*keyword*> field shall be one of the entries from the following list or a keyword provided as
27441 an implementation extension. Keywords consisting entirely of lowercase letters, digits, and
27442 periods are reserved for future standardization. A keyword shall not include an equals sign. (In
27443 the following list, the notations ‘‘file(s)’’ or ‘‘block(s)’’ is used to acknowledge that a keyword
27444 affects the following single file after a *typeflag* **x** extended header, but possibly multiple files after
27445 *typeflag* **g**. Any requirements in the list for *pax* to include a record when in **write** or **copy** mode
27446 shall apply only when such a record has not already been provided through the use of the **−o**
27447 option. When used in **copy** mode, *pax* shall behave as if an archive had been created with
27448 applicable extended header records and then extracted.)

27449 **atime** The file access time for the following file(s), equivalent to the value of the *st_atime*
27450 member of the **stat** structure for a file, as described by the *stat*( ) function. The
27451 access time shall be restored if the process has the appropriate privilege required
27452 to do so. The format of the <*value*> shall be as described in **pax Extended Header**
27453 **File Times** (on page 2916).

27454 **charset** The name of the character set used to encode the data in the following file(s). The
27455 entries in the following table are defined to refer to known standards; additional
27456 names may be agreed on between the originator and recipient.

<value>	Formal Standard
ISO-IR∆646∆1990	ISO / IEC 646: 1990
ISO-IR∆8859∆1∆1998	ISO / IEC 8859-1: 1998
ISO-IR∆8859∆2∆1999	ISO / IEC 8859-2: 1999
ISO-IR∆8859∆3∆1999	ISO / IEC 8859-3: 1999
ISO-IR∆8859∆4∆1998	ISO / IEC 8859-4: 1998
ISO-IR∆8859∆5∆1999	ISO / IEC 8859-5: 1999
ISO-IR∆8859∆6∆1999	ISO / IEC 8859-6: 1999
ISO-IR∆8859∆7∆1987	ISO / IEC 8859-7: 1987
ISO-IR∆8859∆8∆1999	ISO / IEC 8859-8: 1999
ISO-IR∆8859∆9∆1999	ISO / IEC 8859-9: 1999
ISO-IR∆8859∆10∆1998	ISO / IEC 8859-10: 1998
ISO-IR∆8859∆13∆1998	ISO / IEC 8859-13: 1998
ISO-IR∆8859∆14∆1998	ISO / IEC 8859-14: 1998
ISO-IR∆8859∆15∆1999	ISO / IEC 8859-15: 1999
ISO-IR∆10646∆2000	ISO / IEC 10646: 2000
ISO-IR∆10646∆2000∆UTF-8	ISO / IEC 10646, UTF-8 encoding
BINARY	None.

The encoding is included in an extended header for information only; when *pax* is used as described in IEEE Std 1003.1-200x, it shall not translate the file data into any other encoding. The **BINARY** entry indicates unencoded binary data.

When used in **write** or **copy** mode, it is implementation-defined whether *pax* includes a **charset** extended header record for a file.

comment A series of characters used as a comment. All characters in the <*value*> field shall be ignored by *pax*.

ctime The file creation time for the following file(s), equivalent to the value of the *st_ctime* member of the **stat** structure for a file, as described by the *stat*( ) function. The creation time shall be restored if the process has the appropriate privilege required to do so. The format of the <*value*> shall be as described in **pax Extended Header File Times** (on page 2916).

gid The group ID of the group that owns the file, expressed as a decimal number using digits from the ISO / IEC 646: 1991 standard. This record shall override the *gid* field in the following header block(s). When used in **write** or **copy** mode, *pax* shall include a *gid* extended header record for each file whose group ID is greater than 2 097 151 (octal 7 777 777).

gname The group of the file(s), formatted as a group name in the group database. This record shall override the *gid* and *gname* fields in the following header block(s), and any *gid* extended header record. When used in **read**, **copy**, or **list** mode, *pax* shall translate the name from the UTF-8 encoding in the header record to the character set appropriate for the group database on the receiving system. If any of the UTF-8 characters cannot be translated, and if the **–oinvalid=**UTF-8 option is not specified, the results are implementation-defined. When used in **write** or **copy** mode, *pax* shall include a **gname** extended header record for each file whose group name cannot be represented entirely with the letters and digits of the portable character set.

linkpath The pathname of a link being created to another file, of any type, previously archived. This record shall override the *linkname* field in the following **ustar** header block(s). The following **ustar** header block shall determine the type of link created.

If *typeflag* of the following header block is 1, it shall be a hard link. If *typeflag* is 2, it shall be a symbolic link and the **linkpath** value shall be the contents of the symbolic link. The *pax* utility shall translate the name of the link (contents of the symbolic link) from the UTF-8 encoding to the character set appropriate for the local file system. When used in **write** or **copy** mode, *pax* shall include a **linkpath** extended header record for each link whose pathname cannot be represented entirely with the members of the portable character set other than NUL.

mtime The file modification time of the following file(s), equivalent to the value of the *st_mtime* member of the **stat** structure for a file, as described in the *stat*( ) function. This record shall override the *mtime* field in the following header block(s). The modification time shall be restored if the process has the appropriate privilege required to do so. The format of the <*value*> shall be as described in **pax Extended Header File Times** (on page 2916).

path The pathname of the following file(s). This record shall override the *name* and *prefix* fields in the following header block(s). The *pax* utility shall translate the pathname of the file from the UTF-8 encoding to the character set appropriate for the local file system.

When used in **write** or **copy** mode, *pax* shall include a *path* extended header record for each file whose pathname cannot be represented entirely with the members of the portable character set other than NUL.

realtime.*any* The keywords prefixed by ''realtime.'' are reserved for future standardization.

security.*any* The keywords prefixed by ''security.'' are reserved for future standardization.

size The size of the file in octets, expressed as a decimal number using digits from the ISO/IEC 646:1991 standard. This record shall override the *size* field in the following header block(s). When used in **write** or **copy** mode, *pax* shall include a *size* extended header record for each file with a size value greater than 8 589 934 591 (octal 7 777 777 777).

uid The user ID of the file owner, expressed as a decimal number using digits from the ISO/IEC 646:1991 standard. This record shall override the *uid* field in the following header block(s). When used in **write** or **copy** mode, *pax* shall include a *uid* extended header record for each file whose owner ID is greater than 2 097 151 (octal 7 777 777).

uname The owner of the following file(s), formatted as a user name in the user database. This record shall override the *uid* and *uname* fields in the following header block(s), and any *uid* extended header record. When used in **read**, **copy**, or **list** mode, *pax* shall translate the name from the UTF-8 encoding in the header record to the character set appropriate for the user database on the receiving system. If any of the UTF-8 characters cannot be translated, and if the **–oinvalid=** UTF-8 option is not specified, the results are implementation-defined. When used in **write** or **copy** mode, *pax* shall include a **uname** extended header record for each file whose user name cannot be represented entirely with the letters and digits of the portable character set.

If the <*value*> field is zero length, it shall delete any header block field, previously entered extended header value, or global extended header value of the same name.

If a keyword in an extended header record (or in a **–o** option-argument) overrides or deletes a corresponding field in the **ustar** header block, *pax* shall ignore the contents of that header block field.

Unlike the **ustar** header block fields, NULs shall not delimit <*value*>s; all characters within the <*value*> field shall be considered data for the field. None of the length limitations of the **ustar** header block fields in Table 4-13 (on page 2917) shall apply to the extended header records.

pax Extended Header Keyword Precedence

This section describes the precedence in which the various header records and fields and command line options are selected to apply to a file in the archive. When *pax* is used in **read** or **list** modes, it shall determine a file attribute in the following sequence:

1. If –**odelete=***keyword-prefix* is used, the affected attributes shall be determined from step 7., if applicable, or ignored otherwise.
2. If –**o***keyword*:= is used, the affected attributes shall be ignored.
3. If –**o***keyword***:=***value* is used, the affected attribute shall be assigned the value.
4. If there is a *typeflag* **x** extended header record, the affected attribute shall be assigned the <*value*>. When extended header records conflict, the last one given in the header shall take precedence.
5. If –**o***keyword***=***value* is used, the affected attribute shall be assigned the value.
6. If there is a *typeflag* **g** global extended header record, the affected attribute shall be assigned the <*value*>. When global extended header records conflict, the last one given in the global header shall take precedence.
7. Otherwise, the attribute shall be determined from the **ustar** header block.

pax Extended Header File Times

The *pax* utility shall write an **mtime** record for each file in **write** or **copy** modes if the file's modification time cannot be represented exactly in the **ustar** header logical record described in **ustar Interchange Format**. This can occur if the time is out of **ustar** range, or if the file system of the underlying implementation supports non-integer time granularities and the time is not an integer. All of these time records shall be formatted as a decimal representation of the time in seconds since the Epoch. If a period ('.') decimal point character is present, the digits to the right of the point shall represent the units of a subsecond timing granularity, where the first digit is tenths of a second and each subsequent digit is a tenth of the previous digit. In **read** or **copy** mode, the *pax* utility shall truncate the time of a file to the greatest value that is not greater than the input header file time. In **write** or **copy** mode, the *pax* utility shall output a time exactly if it can be represented exactly as a decimal number, and otherwise shall generate only enough digits so that the same time shall be recovered if the file is extracted on a system whose underlying implementation supports the same time granularity.

ustar Interchange Format

A **ustar** archive tape or file shall contain a series of logical records. Each logical record shall be a fixed-size logical record of 512 octets (see below). Although this format may be thought of as being stored on 9-track industry-standard 12.7mm (0.5in) magnetic tape, other types of transportable media are not excluded. Each file archived shall be represented by a header logical record that describes the file, followed by zero or more logical records that give the contents of the file. At the end of the archive file there shall be two 512-octet logical records filled with binary zeros, interpreted as an end-of-archive indicator.

The logical records may be grouped for physical I/O operations, as described under the –**b***blocksize* and –**x ustar** options. Each group of logical records may be written with a single operation equivalent to the *write*( ) function. On magnetic tape, the result of this write shall be a

single tape physical block. The last physical block shall always be the full size, so logical records after the two zero logical records may contain undefined data.

The header logical record shall be structured as shown in the following table. All lengths and offsets are in decimal.

Table 4-13 ustar Header Block

Field Name	Octet Offset	Length (in Octets)
name	0	100
mode	100	8
uid	108	8
gid	116	8
size	124	12
mtime	136	12
chksum	148	8
typeflag	156	1
linkname	157	100
magic	257	6
version	263	2
uname	265	32
gname	297	32
devmajor	329	8
devminor	337	8
prefix	345	155

All characters in the header logical record shall be represented in the coded character set of the ISO/IEC 646:1991 standard. For maximum portability between implementations, names should be selected from characters represented by the portable filename character set as octets with the most significant bit zero. If an implementation supports the use of characters outside of slash and the portable filename character set in names for files, users, and groups, one or more implementation-defined encodings of these characters shall be provided for interchange purposes.

However, the *pax* utility shall never create filenames on the local system that cannot be accessed via the procedures described in IEEE Std 1003.1-200x. If a filename is found on the medium that would create an invalid filename, it is implementation-defined whether the data from the file is stored on the file hierarchy and under what name it is stored. The *pax* utility may choose to ignore these files as long as it produces an error indicating that the file is being ignored.

Each field within the header logical record is contiguous; that is, there is no padding used. Each character on the archive medium shall be stored contiguously.

The fields *magic*, *uname*, and *gname* are character strings each terminated by a NUL character. The fields *name*, *linkname*, and *prefix* are NUL-terminated character strings except when all characters in the array contain non-NUL characters including the last character. The *version* field is two octets containing the characters `"00"` (zero-zero). The *typeflag* contains a single character. All other fields are leading zero-filled octal numbers using digits from the ISO/IEC 646:1991 standard IRV. Each numeric field is terminated by one or more <space> or NUL characters.

The *name* and the *prefix* fields shall produce the pathname of the file. A new pathname shall be formed, if *prefix* is not an empty string (its first character is not NUL), by concatenating *prefix* (up to the first NUL character), a slash character, and *name*; otherwise, *name* is used alone. In either case, *name* is terminated at the first NUL character. If *prefix* begins with a NUL character, it shall be ignored. In this manner, pathnames of at most 256 characters can be supported. If a pathname

does not fit in the space provided, *pax* shall notify the user of the error, and shall not store any part of the file—header or data—on the medium.

The *linkname* field, described below, shall not use the *prefix* to produce a pathname. As such, a *linkname* is limited to 100 characters. If the name does not fit in the space provided, *pax* shall notify the user of the error, and shall not attempt to store the link on the medium.

The *mode* field provides 12 bits encoded in the ISO/IEC 646:1991 standard octal digit representation. The encoded bits shall represent the following values:

Table 4-14 ustar *mode* Field

Bit Value	IEEE Std 1003.1-200x Bit	Description
04 000	S_ISUID	Set UID on execution.
02 000	S_ISGID	Set GID on execution.
01 000	<reserved>	Reserved for future standardization.
00 400	S_IRUSR	Read permission for file owner class.
00 200	S_IWUSR	Write permission for file owner class.
00 100	S_IXUSR	Execute/search permission for file owner class.
00 040	S_IRGRP	Read permission for file group class.
00 020	S_IWGRP	Write permission for file group class.
00 010	S_IXGRP	Execute/search permission for file group class.
00 004	S_IROTH	Read permission for file other class.
00 002	S_IWOTH	Write permission for file other class.
00 001	S_IXOTH	Execute/search permission for file other class.

When appropriate privilege is required to set one of these mode bits, and the user restoring the files from the archive does not have the appropriate privilege, the mode bits for which the user does not have appropriate privilege shall be ignored. Some of the mode bits in the archive format are not mentioned elsewhere in this volume of IEEE Std 1003.1-200x. If the implementation does not support those bits, they may be ignored.

The *uid* and *gid* fields are the user and group ID of the owner and group of the file, respectively.

The *size* field is the size of the file in octets. If the *typeflag* field is set to specify a file to be of type 1 (a link) or 2 (a symbolic link), the *size* field shall be specified as zero. If the *typeflag* field is set to specify a file of type 5 (directory), the *size* field shall be interpreted as described under the definition of that record type. No data logical records are stored for types 1, 2, or 5. If the *typeflag* field is set to 3 (character special file), 4 (block special file), or 6 (FIFO), the meaning of the *size* field is unspecified by this volume of IEEE Std 1003.1-200x, and no data logical records shall be stored on the medium. Additionally, for type 6, the *size* field shall be ignored when reading. If the *typeflag* field is set to any other value, the number of logical records written following the header shall be (*size*+511)/512, ignoring any fraction in the result of the division.

The *mtime* field shall be the modification time of the file at the time it was archived. It is the ISO/IEC 646:1991 standard representation of the octal value of the modification time obtained from the *stat*( ) function.

The *chksum* field shall be the ISO/IEC 646:1991 standard IRV representation of the octal value of the simple sum of all octets in the header logical record. Each octet in the header shall be treated as an unsigned value. These values shall be added to an unsigned integer, initialized to zero, the precision of which is not less than 17 bits. When calculating the checksum, the *chksum* field is treated as if it were all spaces.

The *typeflag* field specifies the type of file archived. If a particular implementation does not recognize the type, or the user does not have appropriate privilege to create that type, the file

shall be extracted as if it were a regular file if the file type is defined to have a meaning for the *size* field that could cause data logical records to be written on the medium (see the previous description for *size*). If conversion to a regular file occurs, the *pax* utility shall produce an error indicating that the conversion took place. All of the *typeflag* fields shall be coded in the ISO/IEC 646:1991 standard IRV:

`0` Represents a regular file. For backward compatibility, a *typeflag* value of binary zero (`'\0'`) should be recognized as meaning a regular file when extracting files from the archive. Archives written with this version of the archive file format create regular files with a *typeflag* value of the ISO/IEC 646:1991 standard IRV `'0'`.

`1` Represents a file linked to another file, of any type, previously archived. Such files are identified by each file having the same device and file serial number. The linked-to name is specified in the *linkname* field with a NUL-character terminator if it is less than 100 octets in length.

`2` Represents a symbolic link. The contents of the symbolic link shall be stored in the *linkname* field.

`3,4` Represent character special files and block special files respectively. In this case the *devmajor* and *devminor* fields shall contain information defining the device, the format of which is unspecified by this volume of IEEE Std 1003.1-200x. Implementations may map the device specifications to their own local specification or may ignore the entry.

`5` Specifies a directory or subdirectory. On systems where disk allocation is performed on a directory basis, the *size* field shall contain the maximum number of octets (which may be rounded to the nearest disk block allocation unit) that the directory may hold. A *size* field of zero indicates no such limiting. Systems that do not support limiting in this manner should ignore the *size* field.

`6` Specifies a FIFO special file. Note that the archiving of a FIFO file archives the existence of this file and not its contents.

`7` Reserved to represent a file to which an implementation has associated some high-performance attribute. Implementations without such extensions should treat this file as a regular file (type 0).

`A-Z` The letters `'A'` to `'Z'`, inclusive, are reserved for custom implementations. All other values are reserved for future versions of IEEE Std 1003.1-200x. |

Attempts to archive a socket using **ustar** interchange format shall produce a diagnostic message. | Handling of other file types is implementation-defined. |

The *magic* field is the specification that this archive was output in this archive format. If this field | contains **ustar** (the five characters from the ISO/IEC 646:1991 standard IRV shown followed by NUL), the *uname* and *gname* fields shall contain the ISO/IEC 646:1991 standard IRV representation of the owner and group of the file, respectively (truncated to fit, if necessary). When the file is restored by a privileged, protection-preserving version of the utility, the user and group databases shall be scanned for these names. If found, the user and group IDs contained within these files shall be used rather than the values contained within the *uid* and *gid* fields.

cpio Interchange Format

The octet-oriented *cpio* archive format shall be a series of entries, each comprising a header that describes the file, the name of the file, and then the contents of the file.

An archive may be recorded as a series of fixed-size blocks of octets. This blocking shall be used only to make physical I/O more efficient. The last group of blocks shall be always at the full size.

For the octet-oriented *cpio* archive format, the individual entry information shall be in the order indicated and described by the following table; see also the **<cpio.h>** header.

Table 4-15 Octet-Oriented *cpio* Archive Entry

Header Field Name	Length (in Octets)	Interpreted as
c_magic	6	Octal number
c_dev	6	Octal number
c_ino	6	Octal number
c_mode	6	Octal number
c_uid	6	Octal number
c_gid	6	Octal number
c_nlink	6	Octal number
c_rdev	6	Octal number
c_mtime	11	Octal number
c_namesize	6	Octal number
c_filesize	11	Octal number
Filename Field Name	**Length**	**Interpreted as**
c_name	*c_namesize*	Pathname string
File Data Field Name	**Length**	**Interpreted as**
c_filedata	*c_filesize*	Data

cpio Header

For each file in the archive, a header as defined previously shall be written. The information in the header fields is written as streams of the ISO/IEC 646:1991 standard characters interpreted as octal numbers. The octal numbers shall be extended to the necessary length by appending the ISO/IEC 646:1991 standard IRV zeros at the most-significant-digit end of the number; the result is written to the most-significant digit of the stream of octets first. The fields shall be interpreted as follows:

c_magic Identify the archive as being a transportable archive by containing the identifying value `"070707"`.

c_dev, *c_ino* Contains values that uniquely identify the file within the archive (that is, no files contain the same pair of *c_dev* and *c_ino* values unless they are links to the same file). The values shall be determined in an unspecified manner.

c_mode Contains the file type and access permissions as defined in the following table.

Table 4-16 Values for *cpio c_mode* Field

File Permissions Name	Value	Indicates
C_IRUSR	000 400	Read by owner
C_IWUSR	000 200	Write by owner
C_IXUSR	000 100	Execute by owner
C_IRGRP	000 040	Read by group
C_IWGRP	000 020	Write by group
C_IXGRP	000 010	Execute by group
C_IROTH	000 004	Read by others
C_IWOTH	000 002	Write by others
C_IXOTH	000 001	Execute by others
C_ISUID	004 000	Set *uid*
C_ISGID	002 000	Set *gid*
C_ISVTX	001 000	Reserved
File Type Name	**Value**	**Indicates**
C_ISDIR	040 000	Directory
C_ISFIFO	010 000	FIFO
C_ISREG	0100 000	Regular file
C_ISLNK	0120 000	Symbolic link
C_ISBLK	060 000	Block special file
C_ISCHR	020 000	Character special file
C_ISSOCK	0140 000	Socket
C_ISCTG	0110 000	Reserved

Directories, FIFOs, symbolic links, and regular files shall be supported on a system conforming to this volume of IEEE Std 1003.1-200x; additional values defined previously are reserved for compatibility with existing systems. Additional file types may be supported; however, such files should not be written to archives intended to be transported to other systems.

c_uid Contains the user ID of the owner.

c_gid Contains the group ID of the group.

c_nlink Contains the number of links referencing the file at the time the archive was created.

c_rdev Contains implementation-defined information for character or block special files.

c_mtime Contains the latest time of modification of the file at the time the archive was created.

c_namesize Contains the length of the pathname, including the terminating NUL character.

c_filesize Contains the length of the file in octets. This shall be the length of the data section following the header structure.

27807 **cpio Filename**

27808 The *c_name* field shall contain the pathname of the file. The length of this field in octets is the
27809 value of *c_namesize*.

27810 If a filename is found on the medium that would create an invalid pathname, it is
27811 implementation-defined whether the data from the file is stored on the file hierarchy and under
27812 what name it is stored.

27813 All characters shall be represented in the ISO/IEC 646:1991 standard IRV. For maximum
27814 portability between implementations, names should be selected from characters represented by
27815 the portable filename character set as octets with the most significant bit zero. If an
27816 implementation supports the use of characters outside the portable filename character set in
27817 names for files, users, and groups, one or more implementation-defined encodings of these
27818 characters shall be provided for interchange purposes. However, the *pax* utility shall never
27819 create filenames on the local system that cannot be accessed via the procedures described
27820 previously in this volume of IEEE Std 1003.1-200x. If a filename is found on the medium that
27821 would create an invalid filename, it is implementation-defined whether the data from the file is
27822 stored on the local file system and under what name it is stored. The *pax* utility may choose to
27823 ignore these files as long as it produces an error indicating that the file is being ignored.

27824 **cpio File Data**

27825 Following *c_name*, there shall be *c_filesize* octets of data. Interpretation of such data occurs in a
27826 manner dependent on the file. If *c_filesize* is zero, no data shall be contained in *c_filedata*.

27827 When restoring from an archive:

27828 • If the user does not have the appropriate privilege to create a file of the specified type, *pax*
27829 shall ignore the entry and write an error message to standard error.

27830 • Only regular files have data to be restored. Presuming a regular file meets any selection
27831 criteria that might be imposed on the format-reading utility by the user, such data shall be
27832 restored.

27833 • If a user does not have appropriate privilege to set a particular mode flag, the flag shall be
27834 ignored. Some of the mode flags in the archive format are not mentioned elsewhere in this
27835 volume of IEEE Std 1003.1-200x. If the implementation does not support those flags, they
27836 may be ignored.

27837 **cpio Special Entries**

27838 FIFO special files, directories, and the trailer shall be recorded with *c_filesize* equal to zero. For
27839 other special files, *c_filesize* is unspecified by this volume of IEEE Std 1003.1-200x. The header for
27840 the next file entry in the archive shall be written directly after the last octet of the file entry
27841 preceding it. A header denoting the filename **TRAILER!!!** shall indicate the end of the archive; |
27842 the contents of octets in the last block of the archive following such a header are undefined. |

27843 **EXIT STATUS**

27844 The following exit values shall be returned:

27845 0 All files were processed successfully.

27846 >0 An error occurred.

CONSEQUENCES OF ERRORS

If *pax* cannot create a file or a link when reading an archive or cannot find a file when writing an archive, or cannot preserve the user ID, group ID, or file mode when the **–p** option is specified, a diagnostic message shall be written to standard error and a non-zero exit status shall be returned, but processing shall continue. In the case where *pax* cannot create a link to a file, *pax* shall not, by default, create a second copy of the file.

If the extraction of a file from an archive is prematurely terminated by a signal or error, *pax* may have only partially extracted the file or (if the **–n** option was not specified) may have extracted a file of the same name as that specified by the user, but which is not the file the user wanted. Additionally, the file modes of extracted directories may have additional bits from the S_IRWXU mask set as well as incorrect modification and access times.

APPLICATION USAGE

The **–p** (privileges) option was invented to reconcile differences between historical *tar* and *cpio* implementations. In particular, the two utilities use **–m** in diametrically opposed ways. The **–p** option also provides a consistent means of extending the ways in which future file attributes can be addressed, such as for enhanced security systems or high-performance files. Although it may seem complex, there are really two modes that are most commonly used:

–p e ‘‘Preserve everything’’. This would be used by the historical superuser, someone with all the appropriate privileges, to preserve all aspects of the files as they are recorded in the archive. The **e** flag is the sum of **o** and **p**, and other implementation-defined attributes.

–p p ‘‘Preserve’’ the file mode bits. This would be used by the user with regular privileges who wished to preserve aspects of the file other than the ownership. The file times are preserved by default, but two other flags are offered to disable these and use the time of extraction.

The one pathname per line format of standard input precludes pathnames containing <newline>s. Although such pathnames violate the portable filename guidelines, they may exist and their presence may inhibit usage of *pax* within shell scripts. This problem is inherited from historical archive programs. The problem can be avoided by listing filename arguments on the command line instead of on standard input.

It is almost certain that appropriate privileges are required for *pax* to accomplish parts of this volume of IEEE Std 1003.1-200x. Specifically, creating files of type block special or character special, restoring file access times unless the files are owned by the user (the **–t** option), or preserving file owner, group, and mode (the **–p** option) all probably require appropriate privileges.

In **read** mode, implementations are permitted to overwrite files when the archive has multiple members with the same name. This may fail if permissions on the first version of the file do not permit it to be overwritten.

The **cpio** and **ustar** formats can only support files up to 8589934592 bytes (8 * 2^30) in size. |

EXAMPLES

The following command:

```
pax –w –f /dev/rmt/1m .
```

copies the contents of the current directory to tape drive 1, medium density (assuming historical System V device naming procedures. The historical BSD device name would be **/dev/rmt9**).

The following commands:

```
mkdir newdir
pax –rw olddir newdir
```

copy the *olddir* directory hierarchy to *newdir*.

```
pax –r –s ',^//*usr//*,,' –f a.pax
```

reads the archive **a.pax**, with all files rooted in **/usr** in the archive extracted relative to the current directory.

Using the option:

```
–o listopt="%M %(atime)T %(size)D %(name)s"
```

overrides the default output description in Standard Output and instead writes:

```
–rw–rw––– Jan 12 15:53 1492 /usr/foo/bar
```

Using the options:

```
–o listopt='%L\t%(size)D\n%.7' \
–o listopt='(name)s\n%(ctime)T\n%T'
```

overrides the default output description in Standard Output and instead writes:

```
/usr/foo/bar –> /tmp   1492
/usr/fo
Jan 12 1991
Jan 31 15:53
```

RATIONALE

The *pax* utility was new, commissioned for the ISO POSIX-2:1993 standard. It represents a peaceful compromise between advocates of the historical *tar* and *cpio* utilities.

A fundamental difference between *cpio* and *tar* was in the way directories were treated. The *cpio* utility did not treat directories differently from other files, and to select a directory and its contents required that each file in the hierarchy be explicitly specified. For *tar*, a directory matched every file in the file hierarchy it rooted.

The *pax* utility offers both interfaces; by default, directories map into the file hierarchy they root. The **–d** option causes *pax* to skip any file not explicitly referenced, as *cpio* historically did. The *tar –style* behavior was chosen as the default because it was believed that this was the more common usage and because *tar* is the more commonly available interface, as it was historically provided on both System V and BSD implementations.

The data interchange format specification in this volume of IEEE Std 1003.1-200x requires that processes with ''appropriate privileges'' shall always restore the ownership and permissions of extracted files exactly as archived. If viewed from the historic equivalence between superuser and ''appropriate privileges'', there are two problems with this requirement. First, users running as superusers may unknowingly set dangerous permissions on extracted files. Second, it is needlessly limiting, in that superusers cannot extract files and own them as superuser unless the archive was created by the superuser. (It should be noted that restoration of ownerships and permissions for the superuser, by default, is historical practice in *cpio*, but not in *tar*.) In order to avoid these two problems, the *pax* specification has an additional ''privilege'' mechanism, the **–p** option. Only a *pax* invocation with the privileges needed, and which has the **–p** option set using the **e** specification character, has the ''appropriate privilege'' to restore full ownership and permission information.

Note also that this volume of IEEE Std 1003.1-200x requires that the file ownership and access permissions shall be set, on extraction, in the same fashion as the *creat*( ) function when provided

the mode stored in the archive. This means that the file creation mask of the user is applied to the file permissions.

Users should note that directories may be created by *pax* while extracting files with permissions that are different from those that existed at the time the archive was created. When extracting sensitive information into a directory hierarchy that no longer exists, users are encouraged to set their file creation mask appropriately to protect these files during extraction.

The table of contents output is written to standard output to facilitate pipeline processing.

An early proposal had hard links displaying for all pathnames. This was removed because it complicates the output of the case where **–v** is not specified and does not match historical *cpio* usage. The hard-link information is available in the **–v** display.

The description of the **–l** option allows implementations to make hard links to symbolic links. IEEE Std 1003.1-200x does not specify any way to create a hard link to a symbolic link, but many implementations provide this capability as an extension. If there are hard links to symbolic links when an archive is created, the implementation is required to archive the hard link in the archive (unless **–H** or **–L** is specified). When in **read** mode and in **copy** mode, implementations supporting hard links to symbolic links should use them when appropriate.

The archive formats inherited from the POSIX.1-1990 standard have certain restrictions that have been brought along from historical usage. For example, there are restrictions on the length of pathnames stored in the archive. When *pax* is used in **copy**(**–rw**) mode (copying directory hierarchies), the ability to use extensions from the **–x***pax* format overcomes these restrictions.

The default *blocksize* value of 5 120 bytes for *cpio* was selected because it is one of the standard block-size values for *cpio*, set when the **–B** option is specified. (The other default block-size value for *cpio* is 512 bytes, and this was considered to be too small.) The default block value of 10 240 bytes for *tar* was selected because that is the standard block-size value for BSD *tar*. The maximum block size of 32 256 bytes (2^{15}–512 bytes) is the largest multiple of 512 bytes that fits into a signed 16-bit tape controller transfer register. There are known limitations in some historical systems that would prevent larger blocks from being accepted. Historical values were chosen to improve compatibility with historical scripts using *dd* or similar utilities to manipulate archives. Also, default block sizes for any file type other than character special file has been deleted from this volume of IEEE Std 1003.1-200x as unimportant and not likely to affect the structure of the resulting archive.

Implementations are permitted to modify the block-size value based on the archive format or the device to which the archive is being written. This is to provide implementations with the opportunity to take advantage of special types of devices, and it should not be used without a great deal of consideration as it almost certainly decreases archive portability.

The intended use of the **–n** option was to permit extraction of one or more files from the archive without processing the entire archive. This was viewed by the standard developers as offering significant performance advantages over historical implementations. The **–n** option in early proposals had three effects; the first was to cause special characters in patterns to not be treated specially. The second was to cause only the first file that matched a pattern to be extracted. The third was to cause *pax* to write a diagnostic message to standard error when no file was found matching a specified pattern. Only the second behavior is retained by this volume of IEEE Std 1003.1-200x, for many reasons. First, it is in general not acceptable for a single option to have multiple effects. Second, the ability to make pattern matching characters act as normal characters is useful for parts of *pax* other than file extraction. Third, a finer degree of control over the special characters is useful because users may wish to normalize only a single special character in a single filename. Fourth, given a more general escape mechanism, the previous behavior of the **–n** option can be easily obtained using the **–s** option or a *sed* script. Finally,

writing a diagnostic message when a pattern specified by the user is unmatched by any file is useful behavior in all cases.

In this version, the –**n** was removed from the **copy** mode synopsis of *pax*; it is inapplicable because there are no pattern operands specified in this mode.

There is another method than *pax* for copying subtrees in IEEE Std 1003.1-200x described as part of the *cp* utility. Both methods are historical practice: *cp* provides a simpler, more intuitive interface, while *pax* offers a finer granularity of control. Each provides additional functionality to the other; in particular, *pax* maintains the hard-link structure of the hierarchy while *cp* does not. It is the intention of the standard developers that the results be similar (using appropriate option combinations in both utilities). The results are not required to be identical; there seemed insufficient gain to applications to balance the difficulty of implementations having to guarantee that the results would be exactly identical.

A single archive may span more than one file. It is suggested that implementations provide informative messages to the user on standard error whenever the archive file is changed.

The –**d** option (do not create intermediate directories not listed in the archive) found in early proposals was originally provided as a complement to the historic –**d** option of *cpio*. It has been deleted.

The –**s** option in early proposals specified a subset of the substitution command from the *ed* utility. As there was no reason for only a subset to be supported, the –**s** option is now compatible with the current *ed* specification. Since the delimiter can be any non-null character, the following usage with single spaces is valid:

```
pax –s " foo bar " ...
```

The –**t** description is worded so as to note that this may cause the access time update caused by some other activity (which occurs while the file is being read) to be overwritten.

The default behavior of *pax* with regard to file modification times is the same as historical implementations of *tar*. It is not the historical behavior of *cpio*.

Because the –**i** option uses **/dev/tty**, utilities without a controlling terminal are not able to use this option.

The –**y** option, found in early proposals, has been deleted because a line containing a single period for the –**i** option has equivalent functionality. The special lines for the –**i** option (a single period and the empty line) are historical practice in *cpio*.

In early drafts, an –**e***charmap* option was included to increase portability of files between systems using different coded character sets. This option was omitted because it was apparent that consensus could not be formed for it. In this version, the use of UTF-8 should be an adequate substitute.

The –**k** option was added to address international concerns about the dangers involved in the character set transformations of –**e** (if the target character set were different from the source, the filenames might be transformed into names matching existing files) and also was made more general to protect files transferred between file systems with different {NAME_MAX} values (truncating a filename on a smaller system might also inadvertently overwrite existing files). As stated, it prevents any overwriting, even if the target file is older than the source. This version adds more granularity of options to solve this problem by introducing the –**oinvalid=** option—specifically the UTF-8 action. (Note that an existing file that is named with a UTF-8 encoding is still subject to overwriting in this case. The –**k** option closes that loophole.)

Some of the file characteristics referenced in this volume of IEEE Std 1003.1-200x might not be supported by some archive formats. For example, neither the *tar* nor *cpio* formats contain the file

access time. For this reason, the **e** specification character has been provided, intended to cause all file characteristics specified in the archive to be retained.

It is required that extracted directories, by default, have their access and modification times and permissions set to the values specified in the archive. This has obvious problems in that the directories are almost certainly modified after being extracted and that directory permissions may not permit file creation. One possible solution is to create directories with the mode specified in the archive, as modified by the *umask* of the user, with sufficient permissions to allow file creation. After all files have been extracted, *pax* would then reset the access and modification times and permissions as necessary.

The list-mode formatting description borrows heavily from the one defined by the *printf* utility. However, since there is no separate operand list to get conversion arguments, the format was extended to allow specifying the name of the conversion argument as part of the conversion specification.

The `T` conversion specifier allows time fields to be displayed in any of the date formats. Unlike the *ls* utility, *pax* does not adjust the format when the date is less than six months in the past. This makes parsing the output more predictable.

The `D` conversion specifier handles the ability to display the major/minor or file size, as with *ls*, by using `%–8(`*`size`*`)D`.

The `L` conversion specifier handles the *ls* display for symbolic links.

Conversion specifiers were added to generate existing known types used for *ls*.

pax Interchange Format

The new POSIX data interchange format was developed primarily to satisfy international concerns that the **ustar** and *cpio* formats did not provide for file, user, and group names encoded in characters outside a subset of the ISO/IEC 646:1991 standard. The standard developers realized that this new POSIX data interchange format should be very extensible because there were other requirements they foresaw in the near future:

- Support international character encodings and locale information
- Support security information (ACLs, and so on)
- Support future file types, such as realtime or contiguous files
- Include data areas for implementation use
- Support systems with words larger than 32 bits and timers with subsecond granularity

The following were not goals for this format because these are better handled by separate utilities or are inappropriate for a portable format:

- Encryption
- Compression
- Data translation between locales and codesets
- *inode* storage

The format chosen to support the goals is an extension of the **ustar** format. Of the two formats previously available, only the **ustar** format was selected for extensions because:

- It was easier to extend in an upward-compatible way. It offered version flags and header block type fields with room for future standardization. The *cpio* format, while possessing a more flexible file naming methodology, could not be extended without breaking some

theoretical implementation or using a dummy filename that could be a legitimate filename.

- Industry experience since the original ''tar wars'' fought in developing the ISO POSIX-1 standard has clearly been in favor of the **ustar** format, which is generally the default output format selected for *pax* implementations on new systems.

The new format was designed with one additional goal in mind: reasonable behavior when an older *tar* or *pax* utility happened to read an archive. Since the POSIX.1-1990 standard mandated that a ''format-reading utility'' had to treat unrecognized *typeflag* values as regular files, this allowed the format to include all the extended information in a pseudo-regular file that preceded each real file. An option is given that allows the archive creator to set up reasonable names for these files on the older systems. Also, the normative text suggests that reasonable file access values be used for this **ustar** header block. Making these header files inaccessible for convenient reading and deleting would not be reasonable. File permissions of 600 or 700 are suggested.

The **ustar** *typeflag* field was used to accommodate the additional functionality of the new format rather than magic or version because the POSIX.1-1990 standard (and, by reference, the previous version of *pax*), mandated the behavior of the format-reading utility when it encountered an unknown *typeflag*, but was silent about the other two fields.

Early proposals of the first revision to IEEE Std 1003.1-200x contained a proposed archive format that was based on compatibility with the standard for tape files (ISO 1001, similar to the format used historically on many mainframes and minicomputers). This format was overly complex and required considerable overhead in volume and header records. Furthermore, the standard developers felt that it would not be acceptable to the community of POSIX developers, so it was later changed to be a format more closely related to historical practice on POSIX systems.

The prefix and name split of pathnames in **ustar** was replaced by the single path extended header record for simplicity.

The concept of a global extended header (*typeflag* **g**) was controversial. If this were applied to an archive being recorded on magnetic tape, a few unreadable blocks at the beginning of the tape could be a serious problem; a utility attempting to extract as many files as possible from a damaged archive could lose a large percentage of file header information in this case. However, if the archive were on a reliable medium, such as a CD-ROM, the global extended header offers considerable potential size reductions by eliminating redundant information. Thus, the text warns against using the global method for unreliable media and provides a method for implanting global information in the extended header for each file, rather than in the *typeflag* **g** records.

No facility for data translation or filtering on a per-file basis is included because the standard developers could not invent an interface that would allow this in an efficient manner. If a filter, such as encryption or compression, is to be applied to all the files, it is more efficient to apply the filter to the entire archive as a single file. The standard developers considered interfaces that would invoke a shell script for each file going into or out of the archive, but the system overhead in this approach was considered to be too high.

One such approach would be to have **filter=** records that give a pathname for an executable. When the program is invoked, the file and archive would be open for standard input/output and all the header fields would be available as environment variables or command-line arguments. The standard developers did discuss such schemes, but they were omitted from IEEE Std 1003.1-200x due to concerns about excessive overhead. Also, the program itself would need to be in the archive if it were to be used portably.

There is currently no portable means of identifying the character set(s) used for a file in the file system. Therefore, *pax* has not been given a mechanism to generate charset records automatically. The only portable means of doing this is for the user to write the archive using the

28120 –**ocharset=***string* command line option. This assumes that all of the files in the archive use the
28121 same encoding. The ‘‘implementation-defined’’ text is included to allow for a system that can
28122 identify the encodings used for each of its files.

28123 The table of standards that accompanies the charset record description is acknowledged to be
28124 very limited. Only a limited number of character set standards is reasonable for maximal
28125 interchange. Any character set is, of course, possible by prior agreement. It was suggested that
28126 EBCDIC be listed, but it was omitted because it is not defined by a formal standard. Formal
28127 standards, and then only those with reasonably large followings, can be included here, simply as
28128 a matter of practicality. The <*value*>s represent names of officially registered character sets in the
28129 format required by the ISO 2375: 1985 standard.

28130 The normal comma or <blank>-separated list rules are not followed in the case of keyword
28131 options to allow ease of argument parsing for *getopts*.

28132 Further information on character encodings is in **pax Archive Character Set Encoding/Decoding**
28133 (on page 2931).

28134 The standard developers have reserved keyword name space for vendor extensions. It is
28135 suggested that the format to be used is:

28136 `VENDOR.keyword`

28137 where *VENDOR* is the name of the vendor or organization in all uppercase letters. It is further
28138 suggested that the keyword following the period be named differently than any of the standard
28139 keywords so that it could be used for future standardization, if appropriate, by omitting the
28140 *VENDOR* prefix.

28141 The <*length*> field in the extended header record was included to make it simpler to step
28142 through the records, even if a record contains an unknown format (to a particular *pax*) with
28143 complex interactions of special characters. It also provides a minor integrity checkpoint within
28144 the records to aid a program attempting to recover files from a damaged archive.

28145 There are no extended header versions of the *devmajor* and *devminor* fields because the
28146 unspecified format **ustar** header field should be sufficient. If they are not, vendor-specific
28147 extended keywords (such as *VENDOR.devmajor*) should be used.

28148 Device and *i*-number labeling of files was not adopted from *cpio*; files are interchanged strictly
28149 on a symbolic name basis, as in **ustar**.

28150 Just as with the **ustar** format descriptions, the new format makes no special arrangements for
28151 multi-volume archives. Each of the *pax* archive types is assumed to be inside a single POSIX file
28152 and splitting that file over multiple volumes (diskettes, tape cartridges, and so on), processing
28153 their labels, and mounting each in the proper sequence are considered to be implementation
28154 details that cannot be described portably.

28155 The *pax* format is intended for interchange, not only for backup on a single (family of) systems. It
28156 is not as densely packed as might be possible for backup:

28157 - It contains information as coded characters that could be coded in binary.

28158 - It identifies extended records with name fields that could be omitted in favor of a fixed-field
28159 layout.

28160 - It translates names into a portable character set and identifies locale-related information,
28161 both of which are probably unnecessary for backup.

28162 The requirements on restoring from an archive are slightly different from the historical wording,
28163 allowing for non-monolithic privilege to bring forward as much as possible. In particular,
28164 attributes such as ‘‘high performance file’’ might be broadly but not universally granted while

set-user-ID or *chown*() might be much more restricted. There is no implication in IEEE Std 1003.1-200x that the security information be honored after it is restored to the file hierarchy, in spite of what might be improperly inferred by the silence on that topic. That is a topic for another standard.

Links are recorded in the fashion described here because a link can be to any file type. It is desirable in general to be able to restore part of an archive selectively and restore all of those files completely. If the data is not associated with each link, it is not possible to do this. However, the data associated with a file can be large, and when selective restoration is not needed, this can be a significant burden. The archive is structured so that files that have no associated data can always be restored by the name of any link name of any link, and the user may choose whether data is recorded with each instance of a file that contains data. The format permits mixing of both types of links in a single archive; this can be done for special needs, and *pax* is expected to interpret such archives on input properly, despite the fact that there is no *pax* option that would force this mixed case on output. (When **–o linkdata** is used, the output must contain the duplicate data, but the implementation is free to include it or omit it when **–o linkdata** is not used.)

The time values are included as extended header records for those implementations needing more than the eleven octal digits allowed by the **ustar** format. Portable file timestamps cannot be negative. If *pax* encounters a file with a negative timestamp in **copy** or **write** mode, it can reject the file, substitute a non-negative timestamp, or generate a non-portable timestamp with a leading '-'. Even though some implementations can support finer file-time granularities than seconds, the normative text requires support only for seconds since the Epoch because the ISO POSIX-1 standard states them that way. The **ustar** format includes only *mtime*; the new format adds *atime* and *ctime* for symmetry. The *atime* access time restored to the file system will be affected by the **–p a** and **–p e** options. The *ctime* creation time (actually *inode* modification time) is described with ''appropriate privilege'' so that it can be ignored when writing to the file system. POSIX does not provide a portable means to change file creation time. Nothing is intended to prevent a non-portable implementation of *pax* from restoring the value.

The *gid*, *size*, and *uid* extended header records were included to allow expansion beyond the sizes specified in the regular *tar* header. New file system architectures are emerging that will exhaust the 12-digit size field. There are probably not many systems requiring more than 8 digits for user and group IDs, but the extended header values were included for completeness, allowing overrides for all of the decimal values in the *tar* header.

The standard developers intended to describe the effective results of *pax* with regard to file ownerships and permissions; implementations are not restricted in timing or sequencing the restoration of such, provided the results are as specified.

Much of the text describing the extended headers refers to use in ''**write** or **copy** modes''. The **copy** mode references are due to the normative text: ''The effect of the copy shall be as if the copied files were written to an archive file and then subsequently extracted ...''. There is certainly no way to test whether *pax* is actually generating the extended headers in **copy** mode, but the effects must be as if it had.

pax Archive Character Set Encoding/Decoding

There is a need to exchange archives of files between systems of different native codesets. Filenames, group names, and user names must be preserved to the fullest extent possible when an archive is read on the receiving platform. Translation of the contents of files is not within the scope of the *pax* utility.

There will also be the need to represent characters that are not available on the receiving platform. These unsupported characters cannot be automatically folded to the local set of characters due to the chance of collisions. This could result in overwriting previous extracted files from the archive or pre-existing files on the system.

For these reasons, the codeset used to represent characters within the extended header records of the *pax* archive must be sufficiently rich to handle all commonly used character sets. The fields requiring translation include, at a minimum, filenames, user names, group names, and link pathnames. Implementations may wish to have localized extended keywords that use non-portable characters.

The standard developers considered the following options:

- The archive creator specifies the well-defined name of the source codeset. The receiver must then recognize the codeset name and perform the appropriate translations to the destination codeset.
- The archive creator includes within the archive the character mapping table for the source codeset used to encode extended header records. The receiver must then read the character mapping table and perform the appropriate translations to the destination codeset.
- The archive creator translates the extended header records in the source codeset into a canonical form. The receiver must then perform the appropriate translations to the destination codeset.

The approach that incorporates the name of the source codeset poses the problem of codeset name registration, and makes the archive useless to *pax* archive decoders that do not recognize that codeset.

Because parts of an archive may be corrupted, the standard developers felt that including the character map of the source codeset was too fragile. The loss of this one key component could result in making the entire archive useless. (The difference between this and the global extended header decision was that the latter has a workaround—duplicating extended header records on unreliable media—but this would be too burdensome for large character set maps.)

Both of the above approaches also put an undue burden on the *pax* archive receiver to handle the cross-product of all source and destination codesets.

To simplify the translation from the source codeset to the canonical form and from the canonical form to the destination codeset, the standard developers decided that the internal representation should be a stateless encoding. A stateless encoding is one where each codepoint has the same meaning, without regard to the decoder being in a specific state. An example of a stateful encoding would be the Japanese Shift-JIS; an example of a stateless encoding would be the ISO / IEC 646: 1991 standard (equivalent to 7-bit ASCII).

For these reasons, the standard developers decided to adopt a canonical format for the representation of file information strings. The obvious, well-endorsed candidate is the ISO / IEC 10646-1: 2000 standard (based in part on Unicode), which can be used to represent the characters of virtually all standardized character sets. The standard developers initially agreed upon using UCS2 (16-bit Unicode) as the internal representation. This repertoire of characters provides a sufficiently rich set to represent all commonly-used codesets.

However, the standard developers found that the 16-bit Unicode representation had some problems. It forced the issue of standardizing byte ordering. The 2-byte length of each character made the extended header records twice as long for the case of strings coded entirely from historical 7-bit ASCII. For these reasons, the standard developers chose the UTF-8 defined in the ISO/IEC 10646-1:2000 standard. This multi-byte representation encodes UCS2 or UCS4 characters reliably and deterministically, eliminating the need for a canonical byte ordering. In addition, NUL octets and other characters possibly confusing to POSIX file systems do not appear, except to represent themselves. It was realized that certain national codesets take up more space after the encoding, due to their placement within the UCS range; it was felt that the usefulness of the encoding of the names outweighs the disadvantage of size increase for file, user, and group names.

The encoding of UTF-8 is as follows:

```
UCS4 Hex Encoding  UTF-8 Binary Encoding

00000000-0000007F  0xxxxxxx
00000080-000007FF  110xxxxx 10xxxxxx
00000800-0000FFFF  1110xxxx 10xxxxxx 10xxxxxx
00010000-001FFFFF  11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
00200000-03FFFFFF  111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
04000000-7FFFFFFF  1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
```

where each 'x' represents a bit value from the character being translated.

ustar Interchange Format

The description of the **ustar** format reflects numerous enhancements over pre-1988 versions of the historical *tar* utility. The goal of these changes was not only to provide the functional enhancements desired, but also to retain compatibility between new and old versions. This compatibility has been retained. Archives written using the old archive format are compatible with the new format.

Implementors should be aware that the previous file format did not include a mechanism to archive directory type files. For this reason, the convention of using a filename ending with slash was adopted to specify a directory on the archive.

The total size of the *name* and *prefix* fields have been set to meet the minimum requirements for {PATH_MAX}. If a pathname will fit within the *name* field, it is recommended that the pathname be stored there without the use of the *prefix* field. Although the name field is known to be too small to contain {PATH_MAX} characters, the value was not changed in this version of the archive file format to retain backward compatibility, and instead the prefix was introduced. Also, because of the earlier version of the format, there is no way to remove the restriction on the *linkname* field being limited in size to just that of the *name* field.

The *size* field is required to be meaningful in all implementation extensions, although it could be zero. This is required so that the data blocks can always be properly counted.

It is suggested that if device special files need to be represented that cannot be represented in the standard format that one of the extension types (**A**-**Z**) be used, and that the additional information for the special file be represented as data and be reflected in the *size* field.

Attempting to restore a special file type, where it is converted to ordinary data and conflicts with an existing filename, need not be specially detected by the utility. If run as an ordinary user, *pax* should not be able to overwrite the entries in, for example, **/dev** in any case (whether the file is converted to another type or not). If run as a privileged user, it should be able to do so, and it would be considered a bug if it did not. The same is true of ordinary data files and similarly

named special files; it is impossible to anticipate the needs of the user (who could really intend to overwrite the file), so the behavior should be predictable (and thus regular) and rely on the protection system as required.

The value 7 in the *typeflag* field is intended to define how contiguous files can be stored in a **ustar** archive. IEEE Std 1003.1-200x does not require the contiguous file extension, but does define a standard way of archiving such files so that all conforming systems can interpret these file types in a meaningful and consistent manner. On a system that does not support extended file types, the *pax* utility should do the best it can with the file and go on to the next.

The file protection modes are those conventionally used by the *ls* utility. This is extended beyond the usage in the ISO POSIX-2 standard to support the ''shared text'' or ''sticky'' bit. It is intended that the conformance document should not document anything beyond the existence of and support of such a mode. Further extensions are expected to these bits, particularly with overloading the set-user-ID and set-group-ID flags.

cpio Interchange Format

The reference to appropriate privilege in the *cpio* format refers to an error on standard output; the **ustar** format does not make comparable statements.

The model for this format was the historical System V *cpio*–**c** data interchange format. This model documents the portable version of the *cpio* format and not the binary version. It has the flexibility to transfer data of any type described within IEEE Std 1003.1-200x, yet is extensible to transfer data types specific to extensions beyond IEEE Std 1003.1-200x (for example, contiguous files). Because it describes existing practice, there is no question of maintaining upward compatibility.

cpio Header

There has been some concern that the size of the *c_ino* field of the header is too small to handle those systems that have very large *inode* numbers. However, the *c_ino* field in the header is used strictly as a hard-link resolution mechanism for archives. It is not necessarily the same value as the *inode* number of the file in the location from which that file is extracted.

The name *c_magic* is based on historical usage.

cpio Filename

For most historical implementations of the *cpio* utility, {PATH_MAX} octets can be used to describe the pathname without the addition of any other header fields (the NUL character would be included in this count). {PATH_MAX} is the minimum value for pathname size, documented as 256 bytes. However, an implementation may use *c_namesize* to determine the exact length of the pathname. With the current description of the **<cpio.h>** header, this pathname size can be as large as a number that is described in six octal digits.

Two values are documented under the *c_mode* field values to provide for extensibility for known file types:

0110 000 Reserved for contiguous files. The implementation may treat the rest of the information for this archive like a regular file. If this file type is undefined, the implementation may create the file as a regular file.

This provides for extensibility of the *cpio* format while allowing for the ability to read old archives. Files of an unknown type may be read as ''regular files'' on some implementations. On a system that does not support extended file types, the *pax* utility should do the best it can with the file and go on to the next.

FUTURE DIRECTIONS

None.

SEE ALSO

cp, *ed*, *getopts*, *printf*, the Base Definitions volume of IEEE Std 1003.1-200x, **<cpio.h>**, the System Interfaces volume of IEEE Std 1003.1-200x, *chown*( ), *creat*( ), *mkdir*( ), *stat*( ), *write*( )

CHANGE HISTORY

First released in Issue 4.

Issue 5

A note is added to the APPLICATION USAGE indicating that the *cpio* and *tar* formats can only support files up to 8 gigabytes in size.

Issue 6

The *pax* utility is aligned with the IEEE P1003.2b draft standard:

- Support has been added for symbolic links in the options and interchange formats.
- A new format has been devised, based on extensions to ustar.
- References to the ‘‘extended’’ *tar* and *cpio* formats derived from the POSIX.1-1990 standard have been changed to remove the ‘‘extended’’ adjective because this could cause confusion with the extended *tar* header added in this revision. (All references to *tar* are actually to **ustar**).

IEEE PASC Interpretation 1003.2 #168 is applied clarifying that *mkdir*( ) and *mkfifo*( ) calls can ignore an [EEXIST] error when extracting an archive.

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

IEEE PASC Interpretation 1003.2 #180 is applied, clarifying how extracted files are created when in **read** mode.

IEEE PASC Interpretation 1003.2 #181 is applied, clarifying the description of the **–t** option. |

IEEE PASC Interpretation 1003.2 #195 is applied. |

IEEE PASC Interpretation 1003.2 #206 is applied, clarifying the handling of links for the **–H**, **–L**, and **–l** options. | |

28369 **NAME**

28370 pr — print files

28371 **SYNOPSIS**

28372 `pr [+`*page*`][–`*column*`][–adFmrt][–e[`*char*`][`*gap*`]][–h` *header*`][–i[`*char*`][`*gap*`]]`
28373 XSI `[–l` *lines*`][–n[`*char*`][`*width*`]][–o` *offset*`][–s[`*char*`]][–w` *width*`][–fp]`
28374 `[`*file*`...]`

28375 **DESCRIPTION**

28376 The *pr* utility is a printing and pagination filter. If multiple input files are specified, each shall be
28377 read, formatted, and written to standard output. By default, the input shall be separated into 66-
28378 line pages, each with:

28379 • A 5-line header that includes the page number, date, time, and the pathname of the file

28380 • A 5-line trailer consisting of blank lines

28381 If standard output is associated with a terminal, diagnostic messages shall be deferred until the
28382 *pr* utility has completed processing.

28383 When options specifying multi-column output are specified, output text columns shall be of
28384 equal width; input lines that do not fit into a text column shall be truncated. By default, text
28385 columns shall be separated with at least one <blank>.

28386 **OPTIONS**

28387 The *pr* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
28388 Utility Syntax Guidelines, except that: the *page* option has a `'+'` delimiter; *page* and *column* can
28389 be multi-digit numbers; some of the option-arguments are optional; and some of the option-
28390 arguments cannot be specified as separate arguments from the preceding option letter. In
28391 particular, the **–s** option does not allow the option letter to be separated from its argument, and
28392 the options **–e**, **–i**, and **–n** require that both arguments, if present, not be separated from the
28393 option letter.

28394 The following options shall be supported. In the following option descriptions, *column*, *lines*,
28395 *offset*, *page*, and *width* are positive decimal integers; *gap* is a non-negative decimal integer.

28396 **+***page* Begin output at page number *page* of the formatted input.

28397 **–***column* Produce multi-column output that is arranged in *column* columns (the default shall
28398 be 1) and is written down each column in the order in which the text is received
28399 from the input file. This option should not be used with **–m**. The options **–e** and **–i**
28400 shall be assumed for multiple text-column output. Whether or not text columns
28401 are produced with identical vertical lengths is unspecified, but a text column shall
28402 never exceed the length of the page (see the **–l** option). When used with **–t**, use the
28403 minimum number of lines to write the output.

28404 **–a** Modify the effect of the **–***column* option so that the columns are filled across the
28405 page in a round-robin order (for example, when *column* is 2, the first input line
28406 heads column 1, the second heads column 2, the third is the second line in column
28407 1, and so on).

28408 **–d** Produce output that is double-spaced; append an extra <newline> following every
28409 <newline> found in the input.

28410 **–e[***char***][***gap***]**
28411 Expand each input <tab> to the next greater column position specified by the
28412 formula *n***gap*+1, where *n* is an integer > 0. If *gap* is zero or is omitted, it shall
28413 default to 8. All <tab>s in the input shall be expanded into the appropriate number
28414 of <space>s. If any non-digit character, *char*, is specified, it shall be used as the

input <tab>.

XSI **–f** Use a <form-feed> for new pages, instead of the default behavior that uses a sequence of <newline>s. Pause before beginning the first page if the standard output is associated with a terminal.

–F Use a <form-feed> for new pages, instead of the default behavior that uses a sequence of <newline>s.

–h *header* Use the string *header* to replace the contents of the *file* operand in the page header.

–i[*char***][***gap***]**
In output, replace multiple <space>s with <tab>s wherever two or more adjacent <space>s reach column positions *gap*+1, 2* *gap*+1, 3* *gap*+1, and so on. If *gap* is zero or is omitted, default tab settings at every eighth column position shall be assumed. If any non-digit character, *char*, is specified, it shall be used as the output <tab>.

–l *lines* Override the 66-line default and reset the page length to *lines*. If *lines* is not greater than the sum of both the header and trailer depths (in lines), the *pr* utility shall suppress both the header and trailer, as if the **–t** option were in effect.

–m Merge files. Standard output shall be formatted so the *pr* utility writes one line from each file specified by a *file* operand, side by side into text columns of equal fixed widths, in terms of the number of column positions. Implementations shall support merging of at least nine *file* operands.

–n[*char***][***width***]**
Provide *width*-digit line numbering (default for *width* shall be 5). The number shall occupy the first *width* column positions of each text column of default output or each line of **–m** output. If *char* (any non-digit character) is given, it shall be appended to the line number to separate it from whatever follows (default for *char* is a <tab>).

–o *offset* Each line of output shall be preceded by offset <space>s. If the **–o** option is not specified, the default offset shall be zero. The space taken is in addition to the output line width (see the **–w** option below).

–p Pause before beginning each page if the standard output is directed to a terminal (*pr* shall write an <alert> to standard error and wait for a <carriage-return> to be read on **/dev/tty**).

–r Write no diagnostic reports on failure to open files.

–s[*char***]** Separate text columns by the single character *char* instead of by the appropriate number of <space>s (default for *char* shall be <tab>).

–t Write neither the five-line identifying header nor the five-line trailer usually supplied for each page. Quit writing after the last line of each file without spacing to the end of the page.

–w *width* Set the width of the line to *width* column positions for multiple text-column output only. If the **–w** option is not specified and the **–s** option is not specified, the default width shall be 72. If the **–w** option is not specified and the **–s** option is specified, the default width shall be 512.

For single column output, input lines shall not be truncated.

28458 **OPERANDS**

28459 The following operand shall be supported:

28460 *file* A pathname of a file to be written. If no *file* operands are specified, or if a *file*
28461 operand is '–', the standard input shall be used.

28462 **STDIN**

28463 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is '–'.
28464 See the INPUT FILES section.

28465 **INPUT FILES**

28466 The input files shall be text files.

28467 The file **/dev/tty** shall be used to read responses required by the –**p** option.

28468 **ENVIRONMENT VARIABLES**

28469 The following environment variables shall affect the execution of *pr*:

28470 *LANG* Provide a default value for the internationalization variables that are unset or null.
28471 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
28472 Internationalization Variables for the precedence of internationalization variables
28473 used to determine the values of locale categories.)

28474 *LC_ALL* If set to a non-empty string value, override the values of all the other
28475 internationalization variables.

28476 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
28477 characters (for example, single-byte as opposed to multi-byte characters in
28478 arguments and input files) and which characters are defined as printable (character
28479 class **print**). Non-printable characters are still written to standard output, but are
28480 not counted for the purpose for column-width and line-length calculations.

28481 *LC_MESSAGES*
28482 Determine the locale that should be used to affect the format and contents of
28483 diagnostic messages written to standard error.

28484 *LC_TIME* Determine the format of the date and time for use in writing header lines.

28485 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

28486 *TZ* Determine the timezone used to calculate date and time strings written in header
28487 lines. If *TZ* is unset or null, an unspecified default timezone shall be used.

28488 **ASYNCHRONOUS EVENTS**

28489 If *pr* receives an interrupt while writing to a terminal, it shall flush all accumulated error
28490 messages to the screen before terminating.

28491 **STDOUT**

28492 The *pr* utility output shall be a paginated version of the original file (or files). This pagination
28493 shall be accomplished using either <form-feed>s or a sequence of <newline>s, as controlled by
28494 XSI the –**F** or –**f** option. Page headers shall be generated unless the –**t** option is specified. The page
28495 headers shall be of the form:

28496 `"\n\n%s %s Page %d\n\n\n", <output of date>, <file>, <page number>`

28497 In the POSIX locale, the <*output of date*> field, representing the date and time of last modification
28498 of the input file (or the current date and time if the input file is standard input), shall be
28499 equivalent to the output of the following command as it would appear if executed at the given
28500 time:

```
date "+%b %e %H:%M %Y"
```

without the trailing <newline>, if the page being written is from standard input. If the page being written is not from standard input, in the POSIX locale, the same format shall be used, but the time used shall be the modification time of the file corresponding to *file* instead of the current time. When the *LC_TIME* locale category is not set to the POSIX locale, a different format and order of presentation of this field may be used.

If the standard input is used instead of a *file* operand, the <*file*> field shall be replaced by a null string.

If the **–h** option is specified, the <*file*> field shall be replaced by the *header* argument.

STDERR

The standard error shall be used for diagnostic messages and for alerting the terminal when **–p** is specified.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

1. Print a numbered list of all files in the current directory:

   ```
   ls –a | pr –n –h "Files in $(pwd)."
   ```

2. Print **file1** and **file2** as a double-spaced, three-column listing headed by ‘‘file list’’:

   ```
   pr –3d –h "file list" file1 file2
   ```

3. Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, . . .:

   ```
   pr –e9 –t <file1 >file2
   ```

RATIONALE

This utility is one of those that does not follow the Utility Syntax Guidelines because of its historical origins. The standard developers could have added new options that obeyed the guidelines (and marked the old options *obsolescent*) or devised an entirely new utility; there are examples of both actions in this volume of IEEE Std 1003.1-200x. Because of its widespread use by historical applications, the standard developers decided to exempt this version of *pr* from many of the guidelines.

Implementations are required to accept option-arguments to the **–h**, **–l**, **–o**, and **–w** options whether presented as part of the same argument or as a separate argument to *pr*, as suggested by the Utility Syntax Guidelines. The **–n** and **–s** options, however, are specified as in historical practice because they are frequently specified without their optional arguments. If a <blank>

were allowed before the option-argument in these cases, a *file* operand could mistakenly be interpreted as an option-argument in historical applications.

The text about the minimum number of lines in multi-column output was included to ensure that a best effort is made in balancing the length of the columns. There are known historical implementations in which, for example, 60-line files are listed by *pr* –2 as one column of 56 lines and a second of 4. Although this is not a problem when a full page with headers and trailers is produced, it would be relatively useless when used with –**t**.

Historical implementations of the *pr* utility have differed in the action taken for the –**f** option. BSD uses it as described here for the –**F** option; System V uses it to change trailing <newline>s on each page to a <form-feed> and, if standard output is a TTY device, sends an <alert> to standard error and reads a line from **/dev/tty** before the first page. There were strong arguments from both sides of this issue concerning historical practice and as a result the –**F** option was added. XSI-conformant systems support the System V historical actions for the –**f** option.

The <*output of date*> field in the –**l** format is specified only for the POSIX locale. As noted, the format can be different in other locales. No mechanism for defining this is present in this volume of IEEE Std 1003.1-200x, as the appropriate vehicle is a message catalog; that is, the format should be specified as a ‘‘message’’.

FUTURE DIRECTIONS

None.

SEE ALSO

expand, *lp*

CHANGE HISTORY

First released in Issue 2.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The –**p** option is added.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

28571 **NAME**

28572 printf — write formatted output

28573 **SYNOPSIS**

28574 `printf format[argument...]`

28575 **DESCRIPTION**

28576 The *printf* utility shall write formatted operands to the standard output. The *argument* operands
28577 shall be formatted under control of the *format* operand.

28578 **OPTIONS**

28579 None.

28580 **OPERANDS**

28581 The following operands shall be supported:

28582 *format* A string describing the format to use to write the remaining operands. See the
28583 EXTENDED DESCRIPTION section.

28584 *argument* The strings to be written to standard output, under the control of *format*. See the
28585 EXTENDED DESCRIPTION section.

28586 **STDIN**

28587 Not used.

28588 **INPUT FILES**

28589 None.

28590 **ENVIRONMENT VARIABLES**

28591 The following environment variables shall affect the execution of *printf*:

28592 *LANG* Provide a default value for the internationalization variables that are unset or null.
28593 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
28594 Internationalization Variables for the precedence of internationalization variables
28595 used to determine the values of locale categories.)

28596 *LC_ALL* If set to a non-empty string value, override the values of all the other
28597 internationalization variables.

28598 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
28599 characters (for example, single-byte as opposed to multi-byte characters in
28600 arguments).

28601 *LC_MESSAGES*
28602 Determine the locale that should be used to affect the format and contents of
28603 diagnostic messages written to standard error.

28604 *LC_NUMERIC*
28605 Determine the locale for numeric formatting. It shall affect the format of numbers
28606 written using the `e`, `E`, `f`, `g`, and `G` conversion specifier characters (if supported).

28607 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

28608 **ASYNCHRONOUS EVENTS**

28609 Default.

28610 **STDOUT**

28611 See the EXTENDED DESCRIPTION section.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

The *format* operand shall be used as the *format* string described in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation with the following exceptions:

1. A <space> in the format string, in any context other than a flag of a conversion specification, shall be treated as an ordinary character that is copied to the output.
2. A `'Δ'` character in the format string shall be treated as a `'Δ'` character, not as a <space>.
3. In addition to the escape sequences shown in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`), `"\ddd"`, where *ddd* is a one, two, or three-digit octal number, shall be written as a byte with the numeric value specified by the octal number.
4. The implementation shall not precede or follow output from the `d` or `u` conversion specifiers with <blank>s not specified by the *format* operand.
5. The implementation shall not precede output from the `o` conversion specifier with zeros not specified by the *format* operand.
6. The `e`, `E`, `f`, `g`, and `G` conversion specifiers need not be supported.
7. An additional conversion specifier character, `b`, shall be supported as follows. The argument shall be taken to be a string that may contain backslash-escape sequences. The following backslash-escape sequences shall be supported:

   — The escape sequences listed in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`), which shall be converted to the characters they represent

   — `"\0ddd"`, where *ddd* is a zero, one, two, or three-digit octal number that shall be converted to a byte with the numeric value specified by the octal number

   — `'\c'`, which shall not be written and shall cause *printf* to ignore any remaining characters in the string operand containing it, any remaining string operands, and any additional characters in the *format* operand

   The interpretation of a backslash followed by any other sequence of characters is unspecified.

   Bytes from the converted string shall be written until the end of the string or the number of bytes indicated by the precision specification is reached. If the precision is omitted, it shall be taken to be infinite, so all bytes up to the end of the converted string shall be written.
8. For each conversion specification that consumes an argument, the next argument operand shall be evaluated and converted to the appropriate type for the conversion as specified below.
9. The *format* operand shall be reused as often as necessary to satisfy the argument operands. Any extra `c` or `s` conversion specifiers shall be evaluated as if a null string argument were supplied; other extra conversion specifications shall be evaluated as if a zero argument were supplied. If the *format* operand contains no conversion specifications and *argument* operands are present, the results are unspecified.

10. If a character sequence in the *format* operand begins with a '%' character, but does not form a valid conversion specification, the behavior is unspecified.

The *argument* operands shall be treated as strings if the corresponding conversion specifier is b, c, or s; otherwise, it shall be evaluated as a C constant, as described by the ISO C standard, with the following extensions:

- A leading plus or minus sign shall be allowed.
- If the leading character is a single-quote or double-quote, the value shall be the numeric value in the underlying codeset of the character following the single-quote or double-quote.

If an argument operand cannot be completely converted into an internal value appropriate to the corresponding conversion specification, a diagnostic message shall be written to standard error and the utility shall not exit with a zero exit status, but shall continue processing any remaining operands and shall write the value accumulated at the time the error was detected to standard output.

It is not considered an error if an argument operand is not completely used for a c or s conversion or if a string operand's first or second character is used to get the numeric value of a character.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The floating-point formatting conversion specifications of *printf*( ) are not required because all arithmetic in the shell is integer arithmetic. The *awk* utility performs floating-point calculations and provides its own **printf** function. The *bc* utility can perform arbitrary-precision floating-point arithmetic, but does not provide extensive formatting capabilities. (This *printf* utility cannot really be used to format *bc* output; it does not support arbitrary precision.) Implementations are encouraged to support the floating-point conversions as an extension.

Note that this *printf* utility, like the *printf*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x on which it is based, makes no special provision for dealing with multi-byte characters when using the %c conversion specification or when a precision is specified in a %b or %s conversion specification. Applications should be extremely cautious using either of these features when there are multi-byte characters in the character set.

No provision is made in this volume of IEEE Std 1003.1-200x which allows field widths and precisions to be specified as '*' since the '*' can be replaced directly in the *format* operand using shell variable substitution. Implementations can also provide this feature as an extension if they so choose.

Hexadecimal character constants as defined in the ISO C standard are not recognized in the *format* operand because there is no consistent way to detect the end of the constant. Octal character constants are limited to, at most, three octal digits, but hexadecimal character constants are only terminated by a non-hex-digit character. In the ISO C standard, the "##" concatenation operator can be used to terminate a constant and follow it with a hexadecimal character to be written. In the shell, concatenation occurs before the *printf* utility has a chance to parse the end of the hexadecimal constant.

The %b conversion specification is not part of the ISO C standard; it has been added here as a portable way to process backslash escapes expanded in string operands as provided by the *echo* utility. See also the APPLICATION USAGE section of *echo* (on page 2534) for ways to use *printf* as a replacement for all of the traditional versions of the *echo* utility.

If an argument cannot be parsed correctly for the corresponding conversion specification, the *printf* utility is required to report an error. Thus, overflow and extraneous characters at the end of an argument being used for a numeric conversion shall be reported as errors.

EXAMPLES

To alert the user and then print and read a series of prompts:

```
printf "\aPlease fill in the following: \nName: "
read name
printf "Phone number: "
read phone
```

To read out a list of right and wrong answers from a file, calculate the percentage correctly, and print them out. The numbers are right-justified and separated by a single <tab>. The percentage is written to one decimal place of accuracy:

```
while read right wrong ; do
    percent=$(echo "scale=1;($right*100)/($right+$wrong)" | bc)
    printf "%2d right\t%2d wrong\t(%s%%)\n" \
        $right $wrong $percent
done < database_file
```

The command:

```
printf "%5d%4d\n" 1 21 321 4321 54321
```

produces:

```
    1  21
  3214321
54321   0
```

Note that the *format* operand is used three times to print all of the given strings and that a '0' was supplied by *printf* to satisfy the last %4d conversion specification.

The *printf* utility is required to notify the user when conversion errors are detected while producing numeric output; thus, the following results would be expected on an implementation with 32-bit twos-complement integers when %d is specified as the *format* operand:

Argument	Standard Output	Diagnostic Output
5a	5	**printf: "5a" not completely converted**
9999999999	2147483647	**printf: "9999999999" arithmetic overflow**
–9999999999	–2147483648	**printf: "–9999999999" arithmetic overflow**
ABC	0	**printf: "ABC" expected numeric value**

The diagnostic message format is not specified, but these examples convey the type of information that should be reported. Note that the value shown on standard output is what would be expected as the return value from the *strtol*( ) function as defined in the System Interfaces volume of IEEE Std 1003.1-200x. A similar correspondence exists between %u and *strtoul*( ) and %e, %f, and %g (if the implementation supports floating-point conversions) and *strtod*( ).

In a locale using the ISO/IEC 646:1991 standard as the underlying codeset, the command:

```
printf "%d\n" 3 +3 –3 \'3 \"+3 "'–3"
```

produces:

3 Numeric value of constant 3

3 Numeric value of constant 3

–3 Numeric value of constant –3

51 Numeric value of the character `'3'` in the ISO/IEC 646:1991 standard codeset

43 Numeric value of the character `'+'` in the ISO/IEC 646:1991 standard codeset

45 Numeric value of the character `'–'` in the ISO/IEC 646:1991 standard codeset

Note that in a locale with multi-byte characters, the value of a character is intended to be the value of the equivalent of the **wchar_t** representation of the character as described in the System Interfaces volume of IEEE Std 1003.1-200x.

RATIONALE

The *printf* utility was added to provide functionality that has historically been provided by *echo*. However, due to irreconcilable differences in the various versions of *echo* extant, the version has few special features, leaving those to this new *printf* utility, which is based on one in the Ninth Edition system.

The EXTENDED DESCRIPTION section almost exactly matches the *printf*( ) function in the ISO C standard, although it is described in terms of the file format notation in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 5, File Format Notation.

FUTURE DIRECTIONS

None.

SEE ALSO

awk, *bc*, *echo*, the System Interfaces volume of IEEE Std 1003.1-200x, *printf*( )

CHANGE HISTORY

First released in Issue 4.

NAME

prs — print an SCCS file (**DEVELOPMENT**)

SYNOPSIS

XSI `prs [–a][–d dataspec][–r[SID]] file...`

XSI `prs [ –e| –l] –c cutoff [–d dataspec] file...`

XSI `prs [ –e| –l] –r[SID][–d dataspec]file...`

DESCRIPTION

The *prs* utility shall write to standard output parts or all of an SCCS file in a user-supplied format.

OPTIONS

The *prs* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines, except that the **–r** option has an optional option-argument. This optional option-argument cannot be presented as a separate argument. The following options shall be supported:

–d *dataspec* Specify the output data specification. The *dataspec* shall be a string consisting of SCCS file *data keywords* (see **Data Keywords** (on page 2946)) interspersed with optional user-supplied text.

–r[*SID***]** Specify the SCCS identification string (SID) of a delta for which information is desired. If no *SID* option-argument is specified, the SID of the most recently created delta shall be assumed. |

–e Request information for all deltas created earlier than and including the delta designated via the **–r** option or the date-time given by the **–c** option.

–l Request information for all deltas created later than and including the delta designated via the **–r** option or the date-time given by the **–c** option.

–c *cutoff* Indicate the *cutoff* date-time, in the form:

YY`[`*MM*`[`*DD*`[`*HH*`[`*MM*`[`*SS*`]]]]]`

For the *YY* component, values in the range [69,99] shall refer to years 1969 to 1999 | inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive. |

Note: It is expected that in a future version of IEEE Std 1003.1-200x the default | century inferred from a 2-digit year will change. (This would apply to all | commands accepting a 2-digit year as input.) |

No changes (deltas) to the SCCS file that were created after the specified *cutoff* | date-time shall be included in the output. Units omitted from the date-time default to their maximum possible values; for example, **–c 7502** is equivalent to **–c 750228235959**.

–a Request writing of information for both removed, that is, *delta type=R* (see *rmdel* (on page 3027)) and existing, that is, *delta type=D*, deltas. If the **–a** option is not specified, information for existing deltas only shall be provided.

OPERANDS

The following operand shall be supported:

file A pathname of an existing SCCS file or a directory. If *file* is a directory, the *prs* utility shall behave as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin

with **s.**) and unreadable files shall be silently ignored.

If exactly one *file* operand appears, and it is '–', the standard input shall be read; each line of the standard input shall be taken to be the name of an SCCS file to be processed. Non-SCCS files and unreadable files shall be silently ignored.

STDIN

The standard input shall be a text file used only when the *file* operand is specified as '–'. Each line of the text file shall be interpreted as an SCCS pathname.

INPUT FILES

Any SCCS files displayed are files of an unspecified format.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *prs*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES.*

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall be a text file whose format is dependent on the data keywords specified with the –**d** option.

Data Keywords

Data keywords specify which parts of an SCCS file shall be retrieved and output. All parts of an SCCS file have an associated data keyword. A data keyword may appear in a *dataspec* multiple times.

The information written by *prs* shall consist of:

1. The user-supplied text
2. Appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*

The format of a data keyword value shall either be simple ('S'), in which keyword substitution is direct, or multi-line ('M').

User-supplied text shall be any text other than recognized data keywords. A <tab> shall be specified by '\t' and <newline> by '\n'. When the –**r** option is not specified, the default *dataspec* shall be:

28856 `:PN::\n\n`

28857 and the following *dataspec* shall be used for each selected delta:

28858 `:Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:`

28859

28860 **SCCS File Data Keywords**

28861 Keyword	Data Item	File Section	Value	Format
28862 **:Dt:**	Delta information	Delta Table	See below*	S
28863 **:DL:**	Delta line statistics	"	**:Li:/:Ld:/:Lu:**	S
28864 **:Li:**	Lines inserted by Delta	"	*nnnnn***	S
28865 **:Ld:**	Lines deleted by Delta	"	*nnnnn***	S
28866 **:Lu:**	Lines unchanged by Delta	"	*nnnnn***	S
28867 **:DT:**	Delta type	"	**D** or **R**	S
28868 **:I:**	SCCS ID string (SID)	"	See below**	S
28869 **:R:**	Release number	"	*nnnn*	S
28870 **:L:**	Level number	"	*nnnn*	S
28871 **:B:**	Branch number	"	*nnnn*	S
28872 **:S:**	Sequence number	"	*nnnn*	S
28873 **:D:**	Date delta created	"	**:Dy:/:Dm:/:Dd:**	S
28874 **:Dy:**	Year delta created	"	*nn*	S
28875 **:Dm:**	Month delta created	"	*nn*	S
28876 **:Dd:**	Day delta created	"	*nn*	S
28877 **:T:**	Time delta created	"	**:Th:::Tm:::Ts:**	S
28878 **:Th:**	Hour delta created	"	*nn*	S
28879 **:Tm:**	Minutes delta created	"	*nn*	S
28880 **:Ts:**	Seconds delta created	"	*nn*	S
28881 **:P:**	Programmer who created Delta	"	*logname*	S
28882 **:DS:**	Delta sequence number	"	*nnnn*	S
28883 **:DP:**	Predecessor Delta sequence number	"	*nnnn*	S
28885 **:DI:**	Sequence number of deltas included, excluded or ignored	"	**:Dn:/:Dx:/:Dg:**	S
28887 **:Dn:**	Deltas included (sequence #)	"	**:DS: :DS: ...**	S
28888 **:Dx:**	Deltas excluded (sequence #)	"	**:DS: :DS: ...**	S
28889 **:Dg:**	Deltas ignored (sequence #)	"	**:DS: :DS: ...**	S
28890 **:MR:**	MR numbers for delta	"	*text*	M
28891 **:C:**	Comments for delta	"	*text*	M
28892 **:UN:**	User names	User Names	*text*	M
28893 **:FL:**	Flag list	Flags	*text*	M
28894 **:Y:**	Module type flag	"	*text*	S
28895 **:MF:**	MR validation flag	"	**yes** or **no**	S
28896 **:MP:**	MR validation program name	"	*text*	S
28897 **:KF:**	Keyword error, warning flag	"	**yes** or **no**	S
28898 **:KV:**	Keyword validation string	"	*text*	S
28899 **:BF:**	Branch flag	"	**yes** or **no**	S
28900 **:J:**	Joint edit flag	"	**yes** or **no**	S
28901 **:LK:**	Locked releases	"	**:R: ...**	S
28902 **:Q:**	User-defined keyword	"	*text*	S

SCCS File Data Keywords				
Keyword	**Data Item**	**File Section**	**Value**	**Format**
:M:	Module name	"	*text*	S
:FB:	Floor boundary	"	**:R:**	S
:CB:	Ceiling boundary	"	**:R:**	S
:Ds:	Default SID	"	**:I:**	S
:ND:	Null delta flag	"	**yes** or **no**	S
:FD:	File descriptive text	Comments	*text*	M
:BD:	Body	Body	*text*	M
:GB:	Gotten body	"	*text*	M
:W:	A form of *what* string	N/A	**:Z::M:\t:I:**	S
:A:	A form of *what* string	N/A	**:Z::Y: :M: :I::Z:**	S
:Z:	*what* string delimiter	N/A	@(#)	S
:F:	SCCS filename	N/A	*text*	S
:PN:	SCCS file pathname	N/A	*text*	S

* **:Dt:=:DT: :I: :D: :T: :P: :DS: :DP:**

** **:R:.:L:.:B:.:S:** if the delta is a branch delta (**:BF:**= =**yes**)
:R:.:L: if the delta is not a branch delta (**:BF:**= =**no**)

*** The line statistics are capped at 99 999. For example, if 100 000 lines were unchanged in a certain revision, **:Lu:** shall produce the value 99 999.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

1. The following example:

```
prs –d "User Names for :F: are:\n:UN:" s.file
```

might write to standard output:

```
User Names for s.file are:
xyz
131
abc
```

2. The following example:

```
prs –d "Delta for pgm :M:: :I: – :D: By :P:" –r s.file
```

might write to standard output:

```
Delta for pgm main.c: 3.7 – 77/12/01 By cas
```

3. As a special case:

```
prs s.file
```

might write to standard output:

```
s.file:
<blank line>
D 1.1 77/12/01 00:00:00 cas 1 000000/00000/00000
MRs:
bl78–12345
bl79–54321
COMMENTS:
this is the comment line for s.file initial delta
<blank line>
```

for each delta table entry of the **D** type. The only option allowed to be used with this special case is the **–a** option.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

admin, *delta*, *get*, *what*

CHANGE HISTORY

First released in Issue 2.

Issue 5

The phrase ‘‘in which keyword substitution is followed by a <newline>’’ is deleted from the end of the second paragraph of **Data Keywords** (on page 2946).

The interpretation of the *YY* component of the **–c** *cutoff* argument is noted.

Issue 6

The normative text is reworded to emphasize the term ‘‘shall’’ for implementation requirements.

The Open Group Base Resolution bwg2001-007 is applied, updating the table in STDOUT with a note that line statistics are capped at 99 999 for the **:Li:**, **:Ld:**, **:Lu:**, and **:DL:** keywords.

The Open Group Interpretation PIN4C.00009 is applied.

NAME

ps — report process status

SYNOPSIS

UP XSI
```
ps [–aA][–defl][–G grouplist][–o format]...[–p proclist][–t termlist]
   [–U userlist][–g grouplist][–n namelist][–u userlist]
```

DESCRIPTION

The *ps* utility shall write information about processes, subject to having the appropriate privileges to obtain information about those processes.

By default, *ps* shall select all processes with the same effective user ID as the current user and the same controlling terminal as the invoker.

OPTIONS

The *ps* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–a Write information for all processes associated with terminals. Implementations may omit session leaders from this list.

–A Write information for all processes.

XSI **–d** Write information for all processes, except session leaders.

XSI **–e** Write information for all processes. (Equivalent to **–A**.)

XSI **–f** Generate a **full** listing. (See the STDOUT section for the contents of a **full** listing.)

XSI **–g** *grouplist* Write information for processes whose session leaders are given in *grouplist*. The application shall ensure that the *grouplist* is a single argument in the form of a <blank> or comma-separated list.

–G *grouplist* Write information for processes whose real group ID numbers are given in *grouplist*. The application shall ensure that the *grouplist* is a single argument in the form of a <blank> or comma-separated list.

XSI **–l** Generate a **long** listing. (See STDOUT for the contents of a **long** listing.)

XSI **–n** *namelist* Specify the name of an alternative system *namelist* file in place of the default. The name of the default file and the format of a *namelist* file are unspecified.

–o *format* Write information according to the format specification given in *format*. This is fully described in the STDOUT section. 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 application shall ensure that the *proclist* is 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 application shall ensure that the *termlist* is a single argument in the form of a <blank> or comma-separated list. Terminal identifiers shall be given in an XSI implementation-defined format. On XSI-conformant systems, they shall be given in one of two forms: the device's filename (for example, **tty04**) or, if the device's filename starts with **tty**, just the identifier following the characters **tty** (for

example, "04").

XSI **–u** *userlist* Write information for processes whose user ID numbers or login names are given in *userlist*. The application shall ensure that the *userlist* is a single argument in the form of a <blank> or comma-separated list. In the listing, the numerical user ID shall be written unless the **–f** option is used, in which case the login name shall be written.

–U *userlist* Write information for processes whose real user ID numbers or login names are given in *userlist*. The application shall ensure that the *userlist* is 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 processes. If any are specified, the default list shall be ignored and *ps* shall select the processes represented by the inclusive OR of all the selection-criteria options.

OPERANDS

None.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *ps*:

COLUMNS Override the system-selected horizontal display line size, used to determine the number of text columns to display. See the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables for valid values and results when it is unset or null.

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

LC_TIME Determine the format and contents of the date and time strings displayed.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

TZ Determine the timezone used to calculate date and time strings displayed. If *TZ* is unset or null, an unspecified default timezone shall be used.

29065 **ASYNCHRONOUS EVENTS**

29066 Default.

29067 **STDOUT**

29068 When the **–o** option is not specified, the standard output format is unspecified.

29069 XSI On XSI-conformant systems, the output format shall be as follows. The column headings and
29070 descriptions of the columns in a *ps* listing are given below. The precise meanings of these fields
29071 are implementation-defined. The letters `'f'` and `'l'` (below) indicate the option (**full** or **long**)
29072 that shall cause the corresponding heading to appear; **all** means that the heading always
29073 appears. Note that these two options determine only what information is provided for a process;
29074 they do not determine which processes are listed.

29075 **F**	(l)	Flags (octal and additive) associated with the process.
29076 **S**	(l)	The state of the process.
29077 **UID**	(f,l)	The user ID number of the process owner; the login name is printed
29078		under the **–f** option.
29079 **PID**	(all)	The process ID of the process; it is possible to kill a process if this
29080		datum is known.
29081 **PPID**	(f,l)	The process ID of the parent process.
29082 **C**	(f,l)	Processor utilization for scheduling.
29083 **PRI**	(l)	The priority of the process; higher numbers mean lower priority.
29084 **NI**	(l)	Nice value; used in priority computation.
29085 **ADDR**	(l)	The address of the process.
29086 **SZ**	(l)	The size in blocks of the core image of the process.
29087 **WCHAN**	(l)	The event for which the process is waiting or sleeping; if blank, the
29088		process is running.
29089 **STIME**	(f)	Starting time of the process.
29090 **TTY**	(all)	The controlling terminal for the process.
29091 **TIME**	(all)	The cumulative execution time for the process.
29092 **CMD**	(all)	The command name; the full command name and its arguments are
29093		written under the **–f** option.

29094 A process that has exited and has a parent, but has not yet been waited for by the parent, shall be
29095 marked **defunct**.

29096 Under the option **–f**, *ps* tries to determine the command name and arguments given when the
29097 process was created by examining memory or the swap area. Failing this, the command name, as
29098 it would appear without the option **–f**, is written in square brackets.

29099 The **–o** option allows the output format to be specified under user control.

29100 The application shall ensure that the format specification is a list of names presented as a single
29101 argument, <blank> or comma-separated. Each variable has a default header. The default header
29102 can be overridden by appending an equals sign and the new text of the header. The rest of the
29103 characters in the argument shall be used as the header text. The fields specified shall be written
29104 in the order specified on the command line, and should be arranged in columns in the output.
29105 The field widths shall be selected by the system to be at least as wide as the header text (default
29106 or overridden value). If the header text is null, such as **–o** *user*=, the field width shall be at least
29107 as wide as the default header text. If all header text fields are null, no header line shall be
29108 written.

29109 The following names are recognized in the POSIX locale:

29110 **ruser** The real user ID of the process. This shall be the textual user ID, if it can be
29111 obtained and the field width permits, or a decimal representation otherwise.

29112 **user** The effective user ID of the process. This shall be the textual user ID, if it can be
29113 obtained and the field width permits, or a decimal representation otherwise.

29114 **rgroup** The real group ID of the process. This shall be the textual group ID, if it can be obtained
29115 and the field width permits, or a decimal representation otherwise.

29116 **group** The effective group ID of the process. This shall be the textual group ID, if it can be
29117 obtained and the field width permits, or a decimal representation otherwise.

29118 **pid** The decimal value of the process ID.

29119 **ppid** The decimal value of the parent process ID.

29120 **pgid** The decimal value of the process group ID.

29121 **pcpu** The ratio of CPU time used recently to CPU time available in the same period,
29122 expressed as a percentage. The meaning of ''recently'' in this context is unspecified. The
29123 CPU time available is determined in an unspecified manner.

29124 **vsz** The size of the process in (virtual) memory in 1 024 byte units as a decimal integer.

29125 **nice** The decimal value of the nice value of the process; see *nice* (on page 2863).

29126 **etime** In the POSIX locale, the elapsed time since the process was started, in the form:

29127 **[**[*dd*−**]***hh*:**]***mm*:*ss*

29128 where *dd* shall represent the number of days, *hh* the number of hours, *mm* the number
29129 of minutes, and *ss* the number of seconds. The *dd* field shall be a decimal integer. The
29130 *hh*, *mm*, and *ss* fields shall be two-digit decimal integers padded on the left with zeros.

29131 **time** In the POSIX locale, the cumulative CPU time of the process in the form:

29132 **[***dd*−**]***hh*:*mm*:*ss*

29133 The *dd*, *hh*, *mm*, and *ss* fields shall be as described in the **etime** specifier.

29134 **tty** The name of the controlling terminal of the process (if any) in the same format used by
29135 the *who* utility.

29136 **comm** The name of the command being executed (*argv*[0] value) as a string.

29137 **args** The command with all its arguments as a string. The implementation may truncate this
29138 value to the field width; it is implementation-defined whether any further truncation
29139 occurs. It is unspecified whether the string represented is a version of the argument list
29140 as it was passed to the command when it started, or is a version of the arguments as
29141 they may have been modified by the application. Applications cannot depend on being
29142 able to modify their argument list and having that modification be reflected in the
29143 output of *ps*.

29144 Any field need not be meaningful in all implementations. In such a case a hyphen (''−'') should
29145 be output in place of the field value.

29146 Only **comm** and **args** shall be allowed to contain <blank>s; all others shall not. Any
29147 implementation-defined variables shall be specified in the system documentation along with the
29148 default header and indicating if the field may contain <blank>s.

29149 The following table specifies the default header to be used in the POSIX locale corresponding to
29150 each format specifier.

29151 **Table 4-17** Variable Names and Default Headers in *ps*

29152

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**		

29160

29161 **STDERR**

29162 The standard error shall be used only for diagnostic messages.

29163 **OUTPUT FILES**

29164 None.

29165 **EXTENDED DESCRIPTION**

29166 None.

29167 **EXIT STATUS**

29168 The following exit values shall be returned:

29169 0 Successful completion.

29170 >0 An error occurred.

29171 **CONSEQUENCES OF ERRORS**

29172 Default.

29173 **APPLICATION USAGE**

29174 Things can change while *ps* is running; the snapshot it gives is only true for an instant, and might
29175 not be accurate by the time it is displayed.

29176 The **args** format specifier is allowed to produce a truncated version of the command arguments.
29177 In some implementations, this information is no longer available when the *ps* utility is executed.

29178 If the field width is too narrow to display a textual ID, the system may use a numeric version.
29179 Normally, the system would be expected to choose large enough field widths, but if a large
29180 number of fields were selected to write, it might squeeze fields to their minimum sizes to fit on
29181 one line. One way to ensure adequate width for the textual IDs is to override the default header
29182 for a field to make it larger than most or all user or group names.

29183 There is no special quoting mechanism for header text. The header text is the rest of the
29184 argument. If multiple header changes are needed, multiple **–o** options can be used, such as:

29185
```
ps –o "user=User Name" –o pid=Process\ ID
```

29186 On some implementations, especially multi-level secure systems, *ps* may be severely restricted
29187 and produce information only about child processes owned by the user.

29188 **EXAMPLES**

29189 The command:

29190
```
ps –o user,pid,ppid=MOM –o args
```

29191 writes at least the following in the POSIX locale:

29192
```
  USER   PID   MOM   COMMAND
helene    34    12   ps –o uid,pid,ppid=MOM –o args
```
29193

The contents of the **COMMAND** field need not be the same in all implementations, due to possible truncation.

RATIONALE

There is very little commonality between BSD and System V implementations of *ps*. Many options conflict or have subtly different usages. The standard developers attempted to select a set of options for the base standard that were useful on a wide range of systems and 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 problematic for users or implementors.

It is recognized that on some implementations, especially multi-level secure systems, *ps* may be nearly useless. The default output has therefore been chosen such that it does not break historical implementations and also 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 this volume of IEEE Std 1003.1-200x were arrived at after considering what concepts were likely to be both reasonably useful to the ''average'' user and had a reasonable chance of being implemented on a wide range of systems. Again it is recognized that not all systems are 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 standard developers 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 environment array of the process. This was deleted because there is no portable way to display it.

The **−A** option is equivalent to the BSD **−g** and the SVID **−e**. Because the two systems differed, a mnemonic compromise was selected.

The **−a** option is described with some optional behavior because the SVID omits session leaders, but BSD does not.

In an early proposal, format specifiers appeared for priority and start time. The former was not defined adequately in this volume of IEEE Std 1003.1-200x and was removed in deference to the defined nice value; the latter because elapsed time was considered to be more useful.

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 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 Shell and Utilities *ps*, the format is the concatenation of each **−o**. Therefore, the user can have an alias or function that defines the beginning of their 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*, *who*, and *write* require that they all use the same format.

The **pcpu** field indicates that the CPU time available is determined in an unspecified manner. This is because it is difficult to express an algorithm that is useful across all possible machine

architectures. Historical counterparts to this value have attempted to show percentage of use in the recent past, such as the preceding minute. Frequently, these values for all processes did not add up to 100%. Implementations are encouraged to provide data in this field to users that will help them identify processes currently affecting the performance of the system.

FUTURE DIRECTIONS

None.

SEE ALSO

kill, *nice*, *renice*

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

NAME

pwd — return working directory name

SYNOPSIS

```
pwd [–L | –P ]
```

DESCRIPTION

The *pwd* utility shall write to standard output an absolute pathname of the current working directory, which does not contain the filenames dot or dot-dot.

OPTIONS

The *pwd* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported by the implementation:

–L If the *PWD* environment variable contains an absolute pathname of the current directory that does not contain the filenames dot or dot-dot, *pwd* shall write this pathname to standard output. Otherwise, the **–L** option shall behave as the **–P** option.

–P The absolute pathname written shall not contain filenames that, in the context of the pathname, refer to files of type symbolic link.

If both **–L** and **–P** are specified, the last one shall apply. If neither **–L** nor **–P** is specified, the *pwd* utility shall behave as if **–L** had been specified.

OPERANDS

None.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *pwd*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

PWD If the **–P** option is in effect, this variable shall be set to an absolute pathname of the current working directory that does not contain any components that specify symbolic links, does not contain any components that are dot, and does not contain any components that are dot-dot. If an application sets or unsets the value of *PWD*, the behavior of *pwd* is unspecified.

29296 **ASYNCHRONOUS EVENTS**
29297 Default.

29298 **STDOUT**
29299 The *pwd* utility output is an absolute pathname of the current working directory:

29300 `"%s\n", <directory pathname>`

29301 **STDERR**
29302 The standard error shall be used only for diagnostic messages.

29303 **OUTPUT FILES**
29304 None.

29305 **EXTENDED DESCRIPTION**
29306 None.

29307 **EXIT STATUS**
29308 The following exit values shall be returned:

29309 0 Successful completion.

29310 >0 An error occurred.

29311 **CONSEQUENCES OF ERRORS**
29312 If an error is detected, output shall not be written to standard output, a diagnostic message shall
29313 be written to standard error, and the exit status is not zero.

29314 **APPLICATION USAGE**
29315 None.

29316 **EXAMPLES**
29317 None.

29318 **RATIONALE**
29319 Some implementations have historically provided *pwd* as a shell special built-in command.

29320 In most utilities, if an error occurs, partial output may be written to standard output. This does
29321 not happen in historical implementations of *pwd*. Because *pwd* is frequently used in historical
29322 shell scripts without checking the exit status, it is important that the historical behavior is
29323 required here; therefore, the CONSEQUENCES OF ERRORS section specifically disallows any
29324 partial output being written to standard output.

29325 **FUTURE DIRECTIONS**
29326 None.

29327 **SEE ALSO**
29328 *cd*, the System Interfaces volume of IEEE Std 1003.1-200x, *getcwd*( )

29329 **CHANGE HISTORY**
29330 First released in Issue 2.

29331 **Issue 6**
29332 The **–P** and **–L** options are added to describe actions relating to symbolic links as specified in the
29333 IEEE P1003.2b draft standard.

29334 **NAME**

29335 qalter — alter batch job

29336 **SYNOPSIS**

29337 BE
```
qalter [–a date_time][–A account_string][–c interval][–e path_name]
    [–h hold_list][–j join_list][–k keep_list][–l resource_list]
    [–m mail_options][–M mail_list][–N name][–o path_name]
    [–p priority][–r y|n][–S path_name_list][–u user_list]
    job_identifier ...
```
29342

29343 **DESCRIPTION**

29344 The attributes of a batch job are altered by a request to the batch server that manages the batch
29345 job. The *qalter* utility is a user-accessible batch client that requests the alteration of the attributes
29346 of one or more batch jobs.

29347 The *qalter* utility shall alter the attributes of those batch jobs, and only those batch jobs, for which
29348 a batch *job_identifier* is presented to the utility.

29349 The *qalter* utility shall alter the attributes of batch jobs in the order in which the batch
29350 *job_identifier*s are presented to the utility.

29351 If the *qalter* utility fails to process a batch *job_identifier* successfully, the utility shall proceed to
29352 process the remaining batch *job_identifier*s, if any.

29353 For each batch *job_identifier* for which the *qalter* utility succeeds, each attribute of the identified
29354 batch job shall be altered as indicated by all the options presented to the utility.

29355 For each identified batch job for which the *qalter* utility fails, the utility shall not alter any
29356 attribute of the batch job.

29357 For each batch job that the *qalter* utility processes, the utility shall not modify any attribute other
29358 than those required by the options and option-arguments presented to the utility.

29359 The *qalter* utility shall alter batch jobs by sending a *Modify Job Request* to the batch server that
29360 manages each batch job. At the time the *qalter* utility exits, it shall have modified the batch job
29361 corresponding to each successfully processed batch *job_identifier*. An attempt to alter the
29362 attributes of a batch job in the RUNNING state is implementation-defined.

29363 **OPTIONS**

29364 The *qalter* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
29365 12.2, Utility Syntax Guidelines.

29366 The following options shall be supported by the implementation:

29367 **–a** *date_time* Redefine the time at which the batch job becomes eligible for execution.

29368 The *date_time* argument shall be in the same form and represent the same time as
29369 for the *touch* utility. The time so represented shall be set into the *Execution_Time*
29370 attribute of the batch job. If the time specified is earlier than the current time, the
29371 **–a** option shall have no effect.

29372 **–A** *account_string*

29373 Redefine the account to which the resource consumption of the batch job should be
29374 charged.

29375 The syntax of the *account_string* option-argument is unspecified.

29376 The *qalter* utility shall set the *Account_Name* attribute of the batch job to the value
29377 of the *account_string* option-argument.

29378 –**c** *interval* Redefine whether the batch job should be checkpointed, and if so, how often.

29379 The *qalter* utility shall accept a value for the interval option-argument that is one of
29380 the following:

29381 `n` No checkpointing is to be performed on the batch batch job
29382 (NO_CHECKPOINT).

29383 `s` Checkpointing is to be performed only when the batch server is shut
29384 down (CHECKPOINT_AT_SHUTDOWN).

29385 `c` Automatic periodic checkpointing is to be performed at the
29386 *Minimum_Cpu_Interval* attribute of the batch queue, in units of CPU
29387 minutes (CHECKPOINT_AT_MIN_CPU_INTERVAL).

29388 `c=`*minutes* Automatic periodic checkpointing is to be performed every *minutes*
29389 of CPU time, or every *Minimum_Cpu_Interval* minutes, whichever is
29390 greater. The *minutes* argument shall conform to the syntax for
29391 unsigned integers and shall be greater than zero.

29392 An implementation may define other checkpoint intervals. The conformance
29393 document for an implementation shall describe any alternative checkpoint
29394 intervals, how they are specified, their internal behavior, and how they affect the
29395 behavior of the utility.

29396 The *qalter* utility shall set the *Checkpoint* attribute of the batch job to the value of the
29397 *interval* option-argument.

29398 –**e** *path_name* Redefine the path to be used for the standard error stream of the batch job.

29399 The *qalter* utility shall accept a *path_name* option-argument that conforms to the
29400 syntax of the *path_name* element defined in the System Interfaces volume of
29401 IEEE Std 1003.1-200x, which can be preceded by a host name element of the form
29402 *hostname*:.

29403 If the *path_name* option-argument constitutes an absolute pathname, the *qalter*
29404 utility shall set the *Error_Path* attribute of the batch job to the value of the
29405 *path_name* option-argument, including the host name element, if present.

29406 If the *path_name* option-argument constitutes a relative pathname and no host
29407 name element is specified, the *qalter* utility shall set the *Error_Path* attribute of the
29408 batch job to the value of the absolute pathname derived by expanding the
29409 *path_name* option-argument relative to the current directory of the process that
29410 executes the *qalter* utility.

29411 If the *path_name* option-argument constitutes a relative pathname and a host name
29412 element is specified, the *qalter* utility shall set the *Error_Path* attribute of the batch
29413 job to the value of the option-argument without expansion.

29414 If the *path_name* option-argument does not include a host name element, the *qalter*
29415 utility shall prefix the pathname in the *Error_Path* attribute with *hostname*:, where
29416 *hostname* is the name of the host upon which the *qalter* utility is being executed.

29417 –**h** *hold_list* Redefine the types of holds, if any, on the batch job. The *qalter* –**h** option shall
29418 accept a value for the *hold_list* option-argument that is a string of alphanumeric
29419 characters in the portable character set.

29420 The *qalter* utility shall accept a value for the *hold_list* option-argument that is a
29421 string of one or more of the characters `'u'`, `'s'`, or `'o'`, or the single character
29422 `'n'`. For each unique character in the *hold_list* option-argument, the *qalter* utility

shall add a value to the *Hold_Types* attribute of the batch job as follows, each representing a different hold type:

u USER

s SYSTEM

o OPERATOR

If any of these characters are duplicated in the *hold_list* option-argument, the duplicates shall be ignored. An existing *Hold_Types* attribute can be cleared by the hold type:

n NO_HOLD

The *qalter* utility shall consider it an error if any hold type other than `'n'` is combined with hold type `'n'`. Strictly conforming applications shall not repeat any of the characters `'u'`, `'s'`, `'o'`, or `'n'` within the *hold_list* option-argument. The *qalter* utility shall permit the repetition of characters, but shall not assign additional meaning to the repeated characters. An implementation may define other hold types. The conformance document for an implementation shall describe any additional hold types, how they are specified, their internal behavior, and how they affect the behavior of the utility.

–**j** *join_list* Redefine which streams of the batch job are to be merged. The *qalter* –**j** option shall accept a value for the *join_list* option-argument that is a string of alphanumeric characters in the portable character set.

The *qalter* utility shall accept a *join_list* option-argument that consists of one or more of the characters `'e'` and `'o'`, or the single character `'n'`.

All of the other batch job output streams specified shall be merged into the output stream represented by the character listed first in the *join_list* option-argument.

For each unique character in the *join_list* option-argument, the *qalter* utility shall add a value to the *Join_Path* attribute of the batch job as follows, each representing a different batch job stream to join:

e The standard error of the batch batch job (JOIN_STD_ERROR).

o The standard output of the batch batch job (JOIN_STD_OUTPUT).

An existing *Join_Path* attribute can be cleared by the join type:

n NO_JOIN

If `'n'` is specified, then no files are joined. The *qalter* utility shall consider it an error if any join type other than `'n'` is combined with join type `'n'`.

Strictly conforming applications shall not repeat any of the characters `'e'`, `'o'`, or `'n'` within the *join_list* option-argument. The *qalter* utility shall permit the repetition of characters, but shall not assign additional meaning to the repeated characters.

An implementation may define other join types. The conformance document for an implementation shall describe any additional batch job streams, how they are specified, their internal behavior, and how they affect the behavior of the utility.

–**k** *keep_list* Redefine which output of the batch job to retain on the execution host.

The *qalter* –**k** option shall accept a value for the *keep_list* option-argument that is a string of alphanumeric characters in the portable character set.

The *qalter* utility shall accept a *keep_list* option-argument that consists of one or more of the characters `'e'` and `'o'` or the single character `'n'`.

For each unique character in the *keep_list* option-argument, the *qalter* utility shall add a value to the *Keep_Files* attribute of the batch job as follows, each representing a different batch job stream to keep:

`e` The standard error of the batch batch job (KEEP_STD_ERROR).

`o` The standard output of the batch batch job (KEEP_STD_OUTPUT).

If both `'e'` and `'o'` are specified, then both files are retained. An existing *Keep_Files* attribute can be cleared by the keep type:

`n` NO_KEEP

If `'n'` is specified, then no files are retained. The *qalter* utility shall consider it an error if any keep type other than `'n'` is combined with keep type `'n'`.

Strictly conforming applications shall not repeat any of the characters `'e'`, `'o'`, or `'n'` within the *keep_list* option-argument. The *qalter* utility shall permit the repetition of characters, but shall not assign additional meaning to the repeated characters. An implementation may define other keep types. The conformance document for an implementation shall describe any additional keep types, how they are specified, their internal behavior, and how they affect the behavior of the utility.

–l *resource_list*

Redefine the resources that are allowed or required by the batch job.

The *qalter* utility shall accept a *resource_list* option-argument that conforms to the following syntax:

```
resource=value[,,resource=value,,...]
```

The *qalter* utility shall set one entry in the value of the *Resource_List* attribute of the batch job for each resource listed in the *resource_list* option-argument.

Because the list of supported resource names might vary by batch server, the *qalter* utility shall rely on the batch server to validate the resource names and associated values. See Section 3.3.3 (on page 2325) for a means of removing *keyword=value* (and *value@keyword*) pairs and other general rules for list-oriented batch job attributes.

–m *mail_options*

Redefine the points in the execution of the batch job at which the batch server is to send mail about a change in the state of the batch job.

The *qalter* **–m** option shall accept a value for the *mail_options* option-argument that is a string of alphanumeric characters in the portable character set.

The *qalter* utility shall accept a value for the *mail_options* option-argument that is a string of one or more of the characters `'e'`, `'b'`, and `'a'`, or the single character `'n'`. For each unique character in the *mail_options* option-argument, the *qalter* utility shall add a value to the *Mail_Users* attribute of the batch job as follows, each representing a different time during the life of a batch job at which to send mail:

`e` MAIL_AT_EXIT

`b` MAIL_AT_BEGINNING

29509 a MAIL_AT_ABORT

29510 If any of these characters are duplicated in the *mail_options* option-argument, the
29511 duplicates shall be ignored.

29512 An existing *Mail_Points* attribute can be cleared by the mail type:

29513 n NO_MAIL

29514 If `'n'` is specified, then mail is not sent. The *qalter* utility shall consider it an error
29515 if any mail type other than `'n'` is combined with mail type `'n'`. Strictly
29516 conforming applications shall not repeat any of the characters `'e'`, `'b'`, `'a'`, or
29517 `'n'` within the *mail_options* option-argument. The *qalter* utility shall permit the
29518 repetition of characters but shall not assign additional meaning to the repeated
29519 characters.

29520 An implementation may define other mail types. The conformance document for
29521 an implementation shall describe any additional mail types, how they are
29522 specified, their internal behavior, and how they affect the behavior of the utility.

29523 **–M** *mail_list* Redefine the list of users to which the batch server that executes the batch job is to
29524 send mail, if the batch server sends mail about the batch job.

29525 The syntax of the *mail_list* option-argument is unspecified. If the implementation
29526 of the *qalter* utility uses a name service to locate users, the utility shall accept the
29527 syntax used by the name service.

29528 If the implementation of the *qalter* utility does not use a name service to locate
29529 users, the implementation shall accept the following syntax for user names:

29530 `mail_address[,,mail_address,,...]`

29531 The interpretation of *mail_address* is implementation-defined.

29532 The *qalter* utility shall set the *Mail_Users* attribute of the batch job to the value of
29533 the *mail_list* option-argument.

29534 **–N** *name* Redefine the name of the batch job.

29535 The *qalter* **–N** option shall accept a value for the *name* option-argument that is a
29536 string of up to 15 alphanumeric characters in the portable character set where the
29537 first character is alphabetic.

29538 The syntax of the *name* option-argument is unspecified.

29539 The *qalter* utility shall set the *Job_Name* attribute of the batch job to the value of the
29540 *name* option-argument.

29541 **–o** *path_name* Redefine the path for the standard output of the batch job.

29542 The *qalter* utility shall accept a *path_name* option-argument that conforms to the
29543 syntax of the *path_name* element defined in the System Interfaces volume of
29544 IEEE Std 1003.1-200x, which can be preceded by a host name element of the form
29545 *hostname*:.

29546 If the *path_name* option-argument constitutes an absolute pathname, the *qalter*
29547 utility shall set the *Output_Path* attribute of the batch job to the value of the
29548 *path_name* option-argument.

29549 If the *path_name* option-argument constitutes a relative pathname and no host
29550 name element is specified, the *qalter* utility shall set the *Output_Path* attribute of the
29551 batch job to the absolute pathname derived by expanding the *path_name* option-

29552 argument relative to the current directory of the process that executes the *qalter*
29553 utility.

29554 If the *path_name* option-argument constitutes a relative pathname and a host name
29555 element is specified, the *qalter* utility shall set the *Output_Path* attribute of the batch
29556 job to option-argument without any expansion of the pathname.

29557 If the *path_name* option-argument does not include a host name element, the *qalter*
29558 utility shall prefix the pathname in the *Output_Path* attribute with *hostname*:, where
29559 *hostname* is the name of the host upon which the *qalter* utility is being executed.

29560 **–p** *priority* Redefine the priority of the batch job.

29561 The *qalter* utility shall accept a value for the priority option-argument that
29562 conforms to the syntax for signed decimal integers, and which is not less than
29563 –1 024 and not greater than 1 023.

29564 The *qalter* utility shall set the *Priority* attribute of the batch job to the value of the
29565 *priority* option-argument.

29566 **–r** `y|n` Redefine whether the batch job is rerunable.

29567 If the value of the option-argument is `'y'`, the *qalter* utility shall set the *Rerunable*
29568 attribute of the batch job to TRUE.

29569 If the value of the option-argument is `'n'`, the *qalter* utility shall set the *Rerunable*
29570 attribute of the batch job to FALSE.

29571 The *qalter* utility shall consider it an error if any character other than `'y'` or `'n'` is
29572 specified in the option-argument.

29573 **–S** *path_name_list*
29574 Redefine the shell that interprets the script at the destination system.

29575 The *qalter* utility shall accept a *path_name_list* option-argument that conforms to
29576 the following syntax:

29577 `pathname[@host][,pathname[@host],...]`

29578 The *qalter* utility shall accept only one pathname that is missing a corresponding
29579 host name. The *qalter* utility shall allow only one pathname per named host.

29580 The *qalter* utility shall add a value to the *Shell_Path_List* attribute of the batch job
29581 for each entry in the *path_name_list* option-argument. See Section 3.3.3 (on page
29582 2325) for a means of removing *keyword=value* (and *value@keyword*) pairs and other
29583 general rules for list-oriented batch job attributes.

29584 **–u** *user_list* Redefine the user name under which the batch job is to run at the destination
29585 system.

29586 The *qalter* utility shall accept a *user_list* option-argument that conforms to the
29587 following syntax:

29588 `username[@host][,,username[@host],,...]`

29589 The *qalter* utility shall accept only one user name that is missing a corresponding
29590 host name. The *qalter* utility shall accept only one user name per named host.

29591 The *qalter* utility shall add a value to the *User_List* attribute of the batch job for each
29592 entry in the *user_list* option-argument. See Section 3.3.3 (on page 2325) for a means
29593 of removing *keyword=value* (and *value@keyword*) pairs and other general rules for
29594 list-oriented batch job attributes.

29595 **OPERANDS**

29596 The *qalter* utility shall accept one or more operands that conform to the syntax for a batch
29597 *job_identifier* (see Section 3.3.1 (on page 2324)).

29598 **STDIN**

29599 Not used.

29600 **INPUT FILES**

29601 None.

29602 **ENVIRONMENT VARIABLES**

29603 The following environment variables shall affect the execution of *qalter*:

29604 *LANG* Provide a default value for the internationalization variables that are unset or null.
29605 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
29606 Internationalization Variables for the precedence of internationalization variables
29607 used to determine the values of locale categories.)

29608 *LC_ALL* If set to a non-empty string value, override the values of all the other
29609 internationalization variables.

29610 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
29611 characters (for example, single-byte as opposed to multi-byte characters in
29612 arguments).

29613 *LC_MESSAGES*
29614 Determine the locale that should be used to affect the format and contents of
29615 diagnostic messages written to standard error.

29616 *LOGNAME* Determine the login name of the user.

29617 *TZ* Determine the timezone used to interpret the *date-time* option-argument. If *TZ* is
29618 unset or null, an unspecified default timezone shall be used.

29619 **ASYNCHRONOUS EVENTS**

29620 Default.

29621 **STDOUT**

29622 None.

29623 **STDERR**

29624 The standard error shall be used only for diagnostic messages. |

29625 **OUTPUT FILES**

29626 None.

29627 **EXTENDED DESCRIPTION**

29628 None.

29629 **EXIT STATUS**

29630 The following exit values shall be returned:

29631 0 Successful completion.

29632 >0 An error occurred.

29633 **CONSEQUENCES OF ERRORS**

29634 In addition to the default behavior, the *qalter* utility shall not be required to write a diagnostic
29635 message to standard error when the error reply received from a batch server indicates that the
29636 batch *job_identifier* does not exist on the server. Whether or not the *qalter* utility attempts to
29637 locate the batch job on other batch servers is implementation-defined.

29638 **APPLICATION USAGE**

29639 None.

29640 **EXAMPLES**

29641 None.

29642 **RATIONALE**

29643 The *qalter* utility allows users to change the attributes of a batch job.

29644 As a means of altering a queued job, the *qalter* utility is superior to deleting and requeuing the
29645 batch job insofar as an altered job retains its place in the queue with some traditional selection
29646 algorithms. In addition, the *qalter* utility is both shorter and simpler than a sequence of *qdel* and
29647 *qsub* utilities.

29648 The result of an attempt on the part of a user to alter a batch job in a RUNNING state is
29649 implementation-defined because a batch job in the RUNNING state will already have opened its
29650 output files and otherwise performed any actions indicated by the options in effect at the time
29651 the batch job began execution.

29652 The options processed by the *qalter* utility are identical to those of the *qsub* utility, with a few
29653 exceptions: **–V**, **–v**, and **–q**. The **–V** and **–v** are inappropriate for the *qalter* utility, since they
29654 capture potentially transient environment information from the submitting process. The **–q**
29655 option would specify a new queue, which would largely negate the previously stated advantage
29656 of using *qalter*; furthermore, the *qmove* utility provides a superior means of moving jobs.

29657 Each of the following paragraphs provides the rationale for a *qalter* option.

29658 Additional rationale concerning these options can be found in the rationale for the *qsub* utility.

29659 The **–a** option allows users to alter the date and time at which a batch job becomes eligible to
29660 run.

29661 The **–A** option allows users to change the account that will be charged for the resources
29662 consumed by the batch job. Support for the **–A** option is mandatory for conforming
29663 implementations of *qalter*, even though support of accounting is optional for servers. Whether or
29664 not to support accounting is left to the implementor of the server, but mandatory support of the
29665 **–A** option assures users of a consistent interface and allows them to control accounting on
29666 servers that support accounting.

29667 The **–c** option allows users to alter the checkpointing interval of a batch job. A checkpointing
29668 system, which is not defined by IEEE Std 1003.1-200x, allows recovery of a batch job at the most
29669 recent checkpoint in the event of a crash. Checkpointing is typically used for jobs that consume
29670 expensive computing time or must meet a critical schedule. Users should be allowed to make
29671 the tradeoff between the overhead of checkpointing and the risk to the timely completion of the
29672 batch job; therefore, this volume of IEEE Std 1003.1-200x provides the checkpointing interval
29673 option. Support for checkpointing is optional for servers.

29674 The **–e** option allows users to alter the name and location of the standard error stream written by
29675 a batch job. However, the path of the standard error stream is meaningless if the value of the
29676 *Join_Path* attribute of the batch job is TRUE.

29677 The **–h** option allows users to set the hold type in the *Hold_Types* attribute of a batch job. The
29678 *qhold* and *qrls* utilities add or remove hold types to the *Hold_Types* attribute, respectively. The **–h**
29679 option has been modified to allow for implementation-defined hold types.

29680 The **–j** option allows users to alter the decision to join (merge) the standard error stream of the
29681 batch job with the standard output stream of the batch job.

The **–l** option allows users to change the resource limits imposed on a batch job.

The **–m** option allows users to modify the list of points in the life of a batch job at which the designated users will receive mail notification.

The **–M** option allows users to alter the list of users who will receive notification about events in the life of a batch job.

The **–N** option allows users to change the name of a batch job.

The **–o** option allows users to alter the name and path to which the standard output stream of the batch job will be written.

The **–P** option allows users to modify the priority of a batch job. Support for priority is optional for batch servers.

The **–r** option allows users to alter the rerunability status of a batch job.

The **–S** option allows users to change the name and location of the shell image that will be invoked to interpret the script of the batch job. This option has been modified to allow a list of shell name and locations associated with different host.

The **–u** option allows users to change the user identifier under which the batch job will execute.

The *job_identifier* operand syntax is provided so that the user can differentiate between the originating and destination (or executing) batch server. These may or may not be the same. The *.server_name* portion identifies the originating batch server, while the *@server* portion identifies the destination batch server.

Historically, the *qalter* utility has been a component of the Network Queuing System (NQS), the existing practice from which this utility has been derived.

FUTURE DIRECTIONS

None.

SEE ALSO

qdel, *qhold*, *qmove*, *qrls*, *qsub*, *touch*, Chapter 3 (on page 2303)

CHANGE HISTORY

Derived from IEEE Std 1003.2d-1994.

Issue 6

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

IEEE PASC Interpretation 1003.2 #182 is applied, clarifying the description of the **–a** option.

29712 **NAME**

29713 qdel — delete batch jobs

29714 **SYNOPSIS**

29715 BE
```
qdel job_identifier ...
```
29716

29717 **DESCRIPTION**

29718 A batch job is deleted by sending a request to the batch server that manages the batch job. A
29719 batch job that has been deleted is no longer subject to management by batch services.

29720 The *qdel* utility is a user-accessible client of batch services that requests the deletion of one or
29721 more batch jobs.

29722 The *qdel* utility shall request a batch server to delete those batch jobs for which a batch
29723 *job_identifier* is presented to the utility.

29724 The *qdel* utility shall delete batch jobs in the order in which their batch *job_identifier*s are
29725 presented to the utility.

29726 If the *qdel* utility fails to process any batch *job_identifier* successfully, the utility shall proceed to
29727 process the remaining batch *job_identifier*s, if any.

29728 The *qdel* utility shall delete each batch job by sending a *Delete Job Request* to the batch server that
29729 manages the batch job.

29730 The *qdel* utility shall not exit until the batch job corresponding to each successfully processed
29731 batch *job_identifier* has been deleted.

29732 **OPTIONS**

29733 None.

29734 **OPERANDS**

29735 The *qdel* utility shall accept one or more operands that conform to the syntax for a batch
29736 *job_identifier* (see Section 3.3.1 (on page 2324)).

29737 **STDIN**

29738 Not used.

29739 **INPUT FILES**

29740 None.

29741 **ENVIRONMENT VARIABLES**

29742 The following environment variables shall affect the execution of *qdel*:

29743 *LANG* Provide a default value for the internationalization variables that are unset or null.
29744 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
29745 Internationalization Variables for the precedence of internationalization variables
29746 used to determine the values of locale categories.)

29747 *LC_ALL* If set to a non-empty string value, override the values of all the other
29748 internationalization variables.

29749 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
29750 characters (for example, single-byte as opposed to multi-byte characters in
29751 arguments).

29752 *LC_MESSAGES*
29753 Determine the locale that should be used to affect the format and contents of
29754 diagnostic messages written to standard error.

LOGNAME Determine the login name of the user.

ASYNCHRONOUS EVENTS

Default.

STDOUT

An implementation of the *qdel* utility may write informative messages to standard output.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

In addition to the default behavior, the *qdel* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qdel* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

The *qdel* utility allows users and administrators to delete jobs.

The *qdel* utility provides functionality that is not otherwise available. For example, the *kill* utility of the operating system does not suffice. First, to use the *kill* utility, the user might have to log in on a remote node, because the *kill* utility does not operate across the network. Second, unlike *qdel*, *kill* cannot remove jobs from queues. Lastly, the arguments of the *qdel* utility are job identifiers rather than process identifiers, and so this utility can be passed the output of the *qselect* utility, thus providing users with a means of deleting a list of jobs.

Because a set of jobs can be selected using the *qselect* utility, the *qdel* utility has not been complicated with options that provide for selection of jobs. Instead, the batch jobs to be deleted are identified individually by their job identifiers.

Historically, the *qdel* utility has been a component of NQS, the existing practice on which it is based. However, the *qdel* utility defined in this volume of IEEE Std 1003.1-200x does not provide an option for specifying a signal number to send to the batch job prior to the killing of the process; that capability has been subsumed by the *qsig* utility.

A discussion was held about the delays of networking and the possibility that the batch server may never respond, due to a down router, down batch server, or other network mishap. The DESCRIPTION records this under the words ‘‘fails to process any job identifier’’. In the broad sense, the network problem is also an error, which causes the failure to process the batch job

identifier.

FUTURE DIRECTIONS

None.

SEE ALSO

kill, *qselect*, *qsig*, Chapter 3 (on page 2303)

CHANGE HISTORY

Derived from IEEE Std 1003.2d-1994.

Issue 6

The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

NAME

qhold — hold batch jobs

SYNOPSIS

BE

```
qhold [–h hold_list] job_identifier ...
```

DESCRIPTION

A hold is placed on a batch job by a request to the batch server that manages the batch job. A batch job that has one or more holds is not eligible for execution. The *qhold* utility is a user-accessible client of batch services that requests one or more types of hold to be placed on one or more batch jobs.

The *qhold* utility shall place holds on those batch jobs for which a batch *job_identifier* is presented to the utility.

The *qhold* utility shall place holds on batch jobs in the order in which their batch *job_identifier*s are presented to the utility. If the *qhold* utility fails to process any batch *job_identifier* successfully, the utility shall proceed to process the remaining batch *job_identifier*s, if any.

The *qhold* utility shall place holds on each batch job by sending a *Hold Job Request* to the batch server that manages the batch job.

The *qhold* utility shall not exit until holds have been placed on the batch job corresponding to each successfully processed batch *job_identifier*.

OPTIONS

The *qhold* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following option shall be supported by the implementation:

–h *hold_list* Define the types of holds to be placed on the batch job.

The *qhold* **–h** option shall accept a value for the *hold_list* option-argument that is a string of alphanumeric characters in the portable character set (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set).

The *qhold* utility shall accept a value for the *hold_list* option-argument that is a string of one or more of the characters `'u'`, `'s'`, or `'o'`, or the single character `'n'`.

For each unique character in the *hold_list* option-argument, the *qhold* utility shall add a value to the *Hold_Types* attribute of the batch job as follows, each representing a different hold type:

u USER

s SYSTEM

o OPERATOR

If any of these characters are duplicated in the *hold_list* option-argument, the duplicates shall be ignored.

An existing *Hold_Types* attribute can be cleared by the following hold type:

n NO_HOLD

The *qhold* utility shall consider it an error if any hold type other than `'n'` is combined with hold type `'n'`.

Strictly conforming applications shall not repeat any of the characters 'u', 's', 'o', or 'n' within the *hold_list* option-argument. The *qhold* utility shall permit the repetition of characters, but shall not assign additional meaning to the repeated characters.

An implementation may define other hold types. The conformance document for an implementation shall describe any additional hold types, how they are specified, their internal behavior, and how they affect the behavior of the utility.

If the **–h** option is not presented to the *qhold* utility, the implementation shall set the *Hold_Types* attribute to USER.

OPERANDS

The *qhold* utility shall accept one or more operands that conform to the syntax for a batch *job_identifier* (see Section 3.3.1 (on page 2324)).

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *qhold*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

LOGNAME Determine the login name of the user.

ASYNCHRONOUS EVENTS

Default.

STDOUT

None.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

In addition to the default behavior, the *qhold* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qhold* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

The *qhold* utility allows users to place a hold on one or more jobs. A hold makes a batch job ineligible for execution.

The *qhold* utility has options that allow the user to specify the type of hold. Should the user wish to place a hold on a set of jobs that meet a selection criteria, such a list of jobs can be acquired using the *qselect* utility.

The **–h** option allows the user to specify the type of hold that is to be placed on the job. This option allows for USER, SYSTEM, OPERATOR, and implementation-defined hold types. The USER and OPERATOR holds are distinct. The batch server that manages the batch job will verify that the user is authorized to set the specified hold for the batch job.

Mail is not required on hold because the administrator has the tools and libraries to build this option if he or she wishes.

Historically, the *qhold* utility has been a part of some existing batch systems, although it has not traditionally been a part of the NQS.

FUTURE DIRECTIONS

None.

SEE ALSO

qselect, Chapter 3 (on page 2303)

CHANGE HISTORY

Derived from IEEE Std 1003.2d-1994.

Issue 6

The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

29927 **NAME**

29928 qmove — move batch jobs

29929 **SYNOPSIS**

29930 BE
```
qmove destination job_identifier ...
```
29931

29932 **DESCRIPTION**

29933 To move a batch job is to remove the batch job from the batch queue in which it resides and
29934 instantiate the batch job in another batch queue. A batch job is moved by a request to the batch
29935 server that manages the batch job. The *qmove* utility is a user-accessible batch client that requests
29936 the movement of one or more batch jobs.

29937 The *qmove* utility shall move those batch jobs, and only those batch jobs, for which a batch
29938 *job_identifier* is presented to the utility.

29939 The *qmove* utility shall move batch jobs in the order in which the corresponding batch
29940 *job_identifier*s are presented to the utility.

29941 If the *qmove* utility fails to process a batch *job_identifier* successfully, the utility shall proceed to
29942 process the remaining batch *job_identifier*s, if any.

29943 The *qmove* utility shall move batch jobs by sending a *Move Job Request* to the batch server that
29944 manages each batch job. The *qmove* utility shall not exit before the batch jobs corresponding to all
29945 successfully processed batch *job_identifier*s have been moved.

29946 **OPTIONS**

29947 None.

29948 **OPERANDS**

29949 The *qmove* utility shall accept one operand that conforms to the syntax for a *destination* (see
29950 Section 3.3.2 (on page 2325)).

29951 The *qmove* utility shall accept one or more operands that conform to the syntax for a batch
29952 *job_identifier* (see Section 3.3.1 (on page 2324)).

29953 **STDIN**

29954 Not used.

29955 **INPUT FILES**

29956 None.

29957 **ENVIRONMENT VARIABLES**

29958 The following environment variables shall affect the execution of *qmove*:

29959 *LANG* Provide a default value for the internationalization variables that are unset or null.
29960 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
29961 Internationalization Variables for the precedence of internationalization variables
29962 used to determine the values of locale categories.)

29963 *LC_ALL* If set to a non-empty string value, override the values of all the other
29964 internationalization variables.

29965 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
29966 characters (for example, single-byte as opposed to multi-byte characters in
29967 arguments).

29968 *LC_MESSAGES*
29969 Determine the locale that should be used to affect the format and contents of
29970 diagnostic messages written to standard error.

LOGNAME Determine the login name of the user.

ASYNCHRONOUS EVENTS

Default.

STDOUT

None.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

In addition to the default behavior, the *qmove* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qmove* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

The *qmove* utility allows users to move jobs between queues.

The alternative to using the *qmove* utility—deleting the batch job and requeuing it—entails considerably more typing.

Since the means of selecting jobs based on attributes has been encapsulated in the *qselect* utility, the only option of the *qmove* utility concerns authorization. The **–u** option provides the user with the convenience of changing the user identifier under which the batch job will execute. Minimalism and consistency has taken precedence over convenience; the **–u** option has been deleted because the equivalent capability exists with the **–u** option of the *qalter* utility.

FUTURE DIRECTIONS

None.

SEE ALSO

qalter, *qselect*, Chapter 3 (on page 2303)

CHANGE HISTORY

Derived from IEEE Std 1003.2d-1994.

30011 **Issue 6**

30012 The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

NAME

qmsg — send message to batch jobs

SYNOPSIS

BE

```
qmsg [–E][–O] message_string job_identifier ...
```

DESCRIPTION

To send a message to a batch job is to request that a server write a message string into one or more output files of the batch job. A message is sent to a batch job by a request to the batch server that manages the batch job. The *qmsg* utility is a user-accessible batch client that requests the sending of messages to one or more batch jobs.

The *qmsg* utility shall write messages into the files of batch jobs by sending a *Job Message Request* to the batch server that manages the batch job. The *qmsg* utility shall not directly write the message into the files of the batch job.

The *qmsg* utility shall send a *Job Message Request* for those batch jobs, and only those batch jobs, for which a batch *job_identifier* is presented to the utility.

The *qmsg* utility shall send *Job Message Request*s for batch jobs in the order in which their batch *job_identifier*s are presented to the utility.

If the *qmsg* utility fails to process any batch *job_identifier* successfully, the utility shall proceed to process the remaining batch *job_identifier*s, if any.

The *qmsg* utility shall not exit before a *Job Message Request* has been sent to the server that manages the batch job that corresponds to each successfully processed batch *job_identifier*.

OPTIONS

The *qmsg* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported by the implementation:

–E Specify that the message is written to the standard error of each batch job.

The *qmsg* utility shall write the message into the standard error of the batch job.

–O Specify that the message is written to the standard output of each batch job.

The *qmsg* utility shall write the message into the standard output of the batch job.

If neither the **–O** nor the **–E** option is presented to the *qmsg* utility, the utility shall write the message into an implementation-defined file. The conformance document for the implementation shall describe the name and location of the implementation-defined file. If both the **–O** and the **–E** options are presented to the *qmsg* utility, then the utility shall write the messages to both standard output and standard error.

OPERANDS

The *qmsg* utility shall accept a minimum of two operands, *message_string* and one or more batch *job_identifier*s.

The *message_string* operand shall be the string to be written to one or more output files of the batch job followed by a <newline>. If the string contains <blank>s, then the application shall ensure that the string is quoted. The *message_string* shall be encoded in the portable character set (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set).

All remaining operands are batch *job_identifier*s that conform to the syntax for a batch *job_identifier* (see Section 3.3.1 (on page 2324)).

30056 **STDIN**

30057 Not used.

30058 **INPUT FILES**

30059 None.

30060 **ENVIRONMENT VARIABLES**

30061 The following environment variables shall affect the execution of *qmsg*:

30062 *LANG* Provide a default value for the internationalization variables that are unset or null.
30063 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
30064 Internationalization Variables for the precedence of internationalization variables
30065 used to determine the values of locale categories.)

30066 *LC_ALL* If set to a non-empty string value, override the values of all the other
30067 internationalization variables.

30068 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
30069 characters (for example, single-byte as opposed to multi-byte characters in
30070 arguments).

30071 *LC_MESSAGES*
30072 Determine the locale that should be used to affect the format and contents of
30073 diagnostic messages written to standard error.

30074 *LOGNAME* Determine the login name of the user.

30075 **ASYNCHRONOUS EVENTS**

30076 Default.

30077 **STDOUT**

30078 None.

30079 **STDERR**

30080 The standard error shall be used only for diagnostic messages.

30081 **OUTPUT FILES**

30082 None.

30083 **EXTENDED DESCRIPTION**

30084 None.

30085 **EXIT STATUS**

30086 The following exit values shall be returned:

30087 0 Successful completion.

30088 >0 An error occurred.

30089 **CONSEQUENCES OF ERRORS**

30090 In addition to the default behavior, the *qmsg* utility shall not be required to write a diagnostic
30091 message to standard error when the error reply received from a batch server indicates that the
30092 batch *job_identifier* does not exist on the server. Whether or not the *qmsg* utility waits to output
30093 the diagnostic message while attempting to locate the job on other servers is implementation-
30094 defined.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

The *qmsg* utility allows users to write messages into the output files of running jobs. Users, including operators and administrators, have a number of occasions when they want to place messages in the output files of a batch job. For example, if a disk that is being used by a batch job is showing errors, the operator might note this in the standard error stream of the batch job.

The options of the *qmsg* utility provide users with the means of placing the message in the output stream of their choice. The default output stream for the message—if the user does not designate an output stream—is implementation-defined, since many implementations will provide, as an extension to this volume of IEEE Std 1003.1-200x, a log file that shows the history of utility execution.

If users wish to send a message to a set of jobs that meet a selection criteria, the *qselect* utility can be used to acquire the appropriate list of job identifiers.

The **–E** option allows users to place the message in the standard error stream of the batch job.

The **–O** option allows users to place the message in the standard output stream of the batch job.

Historically, the *qmsg* utility is an existing practice in the offerings of one or more implementors of an NQS-derived batch system. The utility has been found to be useful enough that it deserves to be included in this volume of IEEE Std 1003.1-200x.

FUTURE DIRECTIONS

None.

SEE ALSO

qselect, Chapter 3 (on page 2303)

CHANGE HISTORY

Derived from IEEE Std 1003.2d-1994.

Issue 6

The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

30124 **NAME**

30125 qrerun — rerun batch jobs

30126 **SYNOPSIS**

30127 BE
```
qrerun job_identifier ...
```
30128

30129 **DESCRIPTION**

30130 To rerun a batch job is to terminate the session leader of the batch job, delete any associated
30131 checkpoint files, and return the batch job to the batch queued state. A batch job is rerun by a
30132 request to the batch server that manages the batch job. The *qrerun* utility is a user-accessible
30133 batch client that requests the rerunning of one or more batch jobs.

30134 The *qrerun* utility shall rerun those batch jobs for which a batch *job_identifier* is presented to the
30135 utility.

30136 The *qrerun* utility shall rerun batch jobs in the order in which their batch *job_identifier*s are
30137 presented to the utility.

30138 If the *qrerun* utility fails to process any batch *job_identifier* successfully, the utility shall proceed
30139 to process the remaining batch *job_identifier*s, if any.

30140 The *qrerun* utility shall rerun batch jobs by sending a *Rerun Job Request* to the batch server that
30141 manages each batch job.

30142 For each successfully processed batch *job_identifier*, the *qrerun* utility shall have rerun the
30143 corresponding batch batch job at the time the utility exits.

30144 **OPTIONS**

30145 None.

30146 **OPERANDS**

30147 The *qrerun* utility shall accept one or more operands that conform to the syntax for a batch
30148 *job_identifier* (see Section 3.3.1 (on page 2324)).

30149 **STDIN**

30150 Not used.

30151 **INPUT FILES**

30152 None.

30153 **ENVIRONMENT VARIABLES**

30154 The following environment variables shall affect the execution of *qrerun*:

30155 *LANG* Provide a default value for the internationalization variables that are unset or null.
30156 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
30157 Internationalization Variables for the precedence of internationalization variables
30158 used to determine the values of locale categories.)

30159 *LC_ALL* If set to a non-empty string value, override the values of all the other
30160 internationalization variables.

30161 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
30162 characters (for example, single-byte as opposed to multi-byte characters in
30163 arguments).

30164 *LC_MESSAGES*
30165 Determine the locale that should be used to affect the format and contents of
30166 diagnostic messages written to standard error.

LOGNAME Determine the login name of the user.

ASYNCHRONOUS EVENTS

Default.

STDOUT

None.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

In addition to the default behavior, the *qrerun* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qrerun* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

The *qrerun* utility allows users to cause jobs in the running state to exit and rerun.

The *qrerun* utility is a new utility, *vis-a-vis* existing practice, that has been defined in this volume of IEEE Std 1003.1-200x to correct user-perceived deficiencies in the existing practice.

FUTURE DIRECTIONS

None.

SEE ALSO

Chapter 3 (on page 2303)

CHANGE HISTORY

Derived from IEEE Std 1003.2d-1994.

Issue 6

The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

30204 **NAME**

30205 qrls — release batch jobs

30206 **SYNOPSIS**

30207 BE
```
qrls [–h hold_list] job_identifier ...
```
30208

30209 **DESCRIPTION**

30210 A batch job might have one or more holds, which prevent the batch job from executing. A batch
30211 job from which all the holds have been removed becomes eligible for execution and is said to
30212 have been released. A batch job hold is removed by sending a request to the batch server that
30213 manages the batch job. The *qrls* utility is a user-accessible client of batch services that requests
30214 holds be removed from one or more batch jobs.

30215 The *qrls* utility shall remove one or more holds from those batch jobs for which a batch
30216 *job_identifier* is presented to the utility.

30217 The *qrls* utility shall remove holds from batch jobs in the order in which their batch *job_identifier*s
30218 are presented to the utility.

30219 If the *qrls* utility fails to process a batch *job_identifier* successfully, the utility shall proceed to
30220 process the remaining batch *job_identifier*s, if any.

30221 The *qrls* utility shall remove holds on each batch job by sending a *Release Job Request* to the batch
30222 server that manages the batch job.

30223 The *qrls* utility shall not exit until the holds have been removed from the batch job
30224 corresponding to each successfully processed batch *job_identifier*.

30225 **OPTIONS**

30226 The *qrls* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
30227 12.2, Utility Syntax Guidelines.

30228 The following option shall be supported by the implementation:

30229 **–h** *hold_list* Define the types of holds to be removed from the batch job.

30230 The *qrls* **–h** option shall accept a value for the *hold_list* option-argument that is a
30231 string of alphanumeric characters in the portable character set (see the Base
30232 Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set).

30233 The *qrls* utility shall accept a value for the *hold_list* option-argument that is a string
30234 of one or more of the characters `'u'`, `'s'`, or `'o'`, or the single character `'n'`.

30235 For each unique character in the *hold_list* option-argument, the *qrls* utility shall add
30236 a value to the *Hold_Types* attribute of the batch job as follows, each representing a
30237 different hold type:

30238 u USER

30239 s SYSTEM

30240 o OPERATOR

30241 If any of these characters are duplicated in the *hold_list* option-argument, the
30242 duplicates shall be ignored.

30243 An existing *Hold_Types* attribute can be cleared by the following hold type:

30244 n NO_HOLD

The *qrls* utility shall consider it an error if any hold type other than 'n' is combined with hold type 'n'.

Strictly conforming applications shall not repeat any of the characters 'u', 's', 'o', or 'n' within the *hold_list* option-argument. The *qrls* utility shall permit the repetition of characters, but shall not assign additional meaning to the repeated characters.

An implementation may define other hold types. The conformance document for an implementation shall describe any additional hold types, how they are specified, their internal behavior, and how they affect the behavior of the utility.

If the –**h** option is not presented to the *qrls* utility, the implementation shall remove the USER hold in the *Hold_Types* attribute.

OPERANDS

The *qrls* utility shall accept one or more operands that conform to the syntax for a batch *job_identifier* (see Section 3.3.1 (on page 2324)).

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *qrls*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

LOGNAME Determine the login name of the user.

ASYNCHRONOUS EVENTS

Default.

STDOUT

None.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

In addition to the default behavior, the *qrls* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qrls* utility waits to output the diagnostic message while attempting to locate the job on other servers is implementation-defined.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

The *qrls* utility allows users, operators, and administrators to remove holds from jobs.

The *qrls* utility does not support any job selection options or wildcard arguments. Users may acquire a list of jobs selected by attributes using the *qselect* utility. For example, a user could select all of their held jobs.

The **–h** option allows the user to specify the type of hold that is to be removed. This option allows for USER, SYSTEM, OPERATOR, and implementation-defined hold types. The batch server that manages the batch job will verify whether the user is authorized to remove the specified hold for the batch job. If more than one type of hold has been placed on the batch job, a user may wish to remove only some of them.

Mail is not required on release because the administrator has the tools and libraries to build this option if required.

The *qrls* utility is a new utility *vis-a-vis* existing practice; it has been defined in this volume of IEEE Std 1003.1-200x as the natural complement to the *qhold* utility.

FUTURE DIRECTIONS

None.

SEE ALSO

qhold, *qselect*, Chapter 3 (on page 2303)

CHANGE HISTORY

Derived from IEEE Std 1003.2d-1994.

Issue 6

The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

NAME

qselect — select batch jobs

SYNOPSIS

BE

```
qselect [–a [op]date_time][–A account_string][–c [op]interval]
    [–h hold_list][–l resource_list][–N name][–p [op]priority]
    [–q destination][–r y|n][–s states][–u user_list]
```

DESCRIPTION

To select a set of batch jobs is to return the batch *job_identifier*s for each batch job that meets a list of selection criteria. A set of batch jobs is selected by a request to a batch server. The *qselect* utility is a user-accessible batch client that requests the selection of batch jobs.

Upon successful completion, the *qselect* utility shall have returned a list of zero or more batch *job_identifier*s that meet the criteria specified by the options and option-arguments presented to the utility.

The *qselect* utility shall select batch jobs by sending a *Select Jobs Request* to a batch server. The *qselect* utility shall not exit until the server replies to each request generated.

For each option presented to the *qselect* utility, the utility shall restrict the set of selected batch jobs as described in the OPTIONS section.

The *qselect* utility shall not restrict selection of batch jobs except by authorization and as required by the options presented to the utility.

When an option is specified with a mandatory or optional *op* component to the option-argument, then *op* shall specify a relation between the value of a certain batch job attribute and the *value* component of the option-argument. If an *op* is allowable on an option, then the description of the option letter indicates the *op* as either mandatory or optional. Acceptable strings for the *op* component, and the relation the string indicates, are shown in the following list:

`.eq.` The value represented by the attribute of the batch job is equal to the value represented by the option-argument.

`.ge.` The value represented by the attribute of the batch job is greater than or equal to the value represented by the option-argument.

`.gt.` The value represented by the attribute of the batch job is greater than the value represented by the option-argument.

`.lt.` The value represented by the attribute of the batch job is less than the value represented by the option-argument.

`.le.` The value represented by the attribute of the batch job is less than or equal to the value represented by the option-argument.

`.ne.` The value represented by the attribute of the batch job is not equal to the value represented by the option-argument.

OPTIONS

The *qselect* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported by the implementation:

–a [*op***]***date_time*
Restrict selection to a specific time, or a range of times.

The *qselect* utility shall select only batch jobs for which the value of the *Execution_Time* attribute is related to the Epoch equivalent of the local time expressed by the value of the *date_time* component of the option-argument in the manner indicated by the value of the *op* component of the option-argument.

The *qselect* utility shall accept a *date_time* component of the option-argument that conforms to the syntax of the *time* operand of the *touch* utility.

If the *op* component of the option-argument is not presented to the *qselect* utility, the utility shall select batch jobs for which the *Execution_Time* attribute is equal to the *date_time* component of the option-argument.

When comparing times, the *qselect* utility shall use the following definitions for the *op* component of the option-argument:

`.eq.` The time represented by value of the *Execution_Time* attribute of the batch job is equal the time represented by the *date_time* component of the option-argument.

`.ge.` The time represented by value of the *Execution_Time* attribute of the batch job is after or equal to the time represented by the *date_time* component of the option-argument.

`.gt.` The time represented by value of the *Execution_Time* attribute of the batch job is after the time represented by the *date_time* component of the option-argument.

`.lt.` The time represented by value of the *Execution_Time* attribute of the batch job is before the time represented by the *date_time* component of the option-argument.

`.le.` The time represented by value of the *Execution_Time* attribute of the batch job is before or equal to the time represented by the *date_time* component of the option-argument.

`.ne.` The time represented by value of the *Execution_Time* attribute of the batch job is not equal to the time represented by the *date_time* component of the option-argument.

The *qselect* utility shall accept the defined character strings for the *op* component of the option-argument.

–A *account_string*

Restrict selection to the batch jobs charging a specified account.

The *qselect* utility shall select only batch jobs for which the value of the *Account_Name* attribute of the batch job matchs the value of the *account_string* option-argument.

The syntax of the *account_string* option-argument is unspecified.

–c [*op***]***interval*

Restrict selection to batch jobs within a range of checkpoint intervals.

The *qselect* utility shall select only batch jobs for which the value of the *Checkpoint* attribute relates to the value of the *interval* component of the option-argument in the manner indicated by the value of the *op* component of the option-argument.

If the *op* component of the option-argument is omitted, the *qselect* utility shall select batch jobs for which the value of the *Checkpoint* attribute is equal to the value

of the *interval* component of the option-argument.

When comparing checkpoint intervals, the *qselect* utility shall use the following definitions for the *op* component of the option-argument:

`.eq.` The value of the *Checkpoint* attribute of the batch job equals the value of the *interval* component of the option-argument.

`.ge.` The value of the *Checkpoint* attribute of the batch job is greater than or equal to the value of the *interval* component option-argument.

`.gt.` The value of the *Checkpoint* attribute of the batch job is greater than the value of the *interval* component option-argument.

`.lt.` The value of the *Checkpoint* attribute of the batch job is less than the value of the *interval* component option-argument.

`.le.` The value of the *Checkpoint* attribute of the batch job is less than or equal to the value of the *interval* component option-argument.

`.ne.` The value of the *Checkpoint* attribute of the batch job does not equal the value of the *interval* component option-argument.

The *qselect* utility shall accept the defined character strings for the *op* component of the option-argument.

The ordering relationship for the values of the interval option-argument is defined to be:

`'n' .gt. 's' .gt. 'c=`*`minutes`*`' .ge. 'c'`

When comparing *Checkpoint* attributes with an interval having the value of the single character `'u'`, only equality or inequality are valid comparisons.

–h *hold_list* Restrict selection to batch jobs that have a specific type of hold.

The *qselect* utility shall select only batch jobs for which the value of the *Hold_Types* attribute matches the value of the *hold_list* option-argument.

The *qselect* **–h** option shall accept a value for the *hold_list* option-argument that is a string of alphanumeric characters in the portable character set (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set).

The *qselect* utility shall accept a value for the *hold_list* option-argument that is a string of one or more of the characters `'u'`, `'s'`, or `'o'`, or the single character `'n'`.

Each unique character in the *hold_list* option-argument of the *qselect* utility is defined as follows, each representing a different hold type:

`u` USER

`s` SYSTEM

`o` OPERATOR

If any of these characters are duplicated in the *hold_list* option-argument, the duplicates shall be ignored.

The *qselect* utility shall consider it an error if any hold type other than `'n'` is combined with hold type `'n'`.

Strictly conforming applications shall not repeat any of the characters `'u'`, `'s'`, `'o'`, or `'n'` within the *hold_list* option-argument. The *qselect* utility shall permit the repetition of characters, but shall not assign additional meaning to the repeated characters.

An implementation may define other hold types. The conformance document for an implementation shall describe any additional hold types, how they are specified, their internal behavior, and how they affect the behavior of the utility.

–l *resource_list*

Restrict selection to batch jobs with specified resource limits and attributes.

The *qselect* utility shall accept a *resource_list* option-argument with the following syntax:

resource_name op value **[**,,*resource_name op value*,, ...**]**

When comparing resource values, the *qselect* utility shall use the following definitions for the *op* component of the option-argument:

`.eq.` The value of the resource of the same name in the *Resource_List* attribute of the batch job equals the value of the value component of the option-argument.

`.ge.` The value of the resource of the same name in the *Resource_List* attribute of the batch job is greater than or equal to the value of the *value* component of the option-argument.

`.gt.` The value of the resource of the same name in the *Resource_List* attribute of the batch job is greater than the value of the value component of the option-argument.

`.lt.` The value of the resource of the same name in the *Resource_List* attribute of the batch job is less than the value of the value component of the option-argument.

`.ne.` The value of the resource of the same name in the *Resource_List* attribute of the batch job does not equal the value of the value component of the option-argument.

`.le.` The value of the resource of the same name in the *Resource_List* attribute of the batch job is less than or equal to the value of the *value* component of the option-argument.

When comparing the limit of a *Resource_List* attribute with the *value* component of the option-argument, if the limit, the value, or both are non-numeric, only equality or inequality are valid comparisons.

The *qselect* utility shall select only batch jobs for which the values of the *resource_name*s listed in the *resource_list* option-argument match the corresponding limits of the *Resource_List* attribute of the batch job.

Limits of *resource_name*s present in the *Resource_List* attribute of the batch job that have no corresponding values in the *resource_list* option-argument shall not be considered when selecting batch jobs.

–N *name* Restrict selection to batch jobs with a specified name.

The *qselect* utility shall select only batch jobs for which the value of the *Job_Name* attribute matches the value of the *name* option-argument. The string specified in

the *name* option-argument shall be passed, uninterpreted, to the server. This allows an implementation to match ''wildcard'' patterns against batch job names.

An implementation shall describe in the conformance document the format it supports for matching against the *Job_Name* attribute.

–p [*op***]***priority*

Restrict selection to batch jobs of the specified priority or range of priorities.

The *qselect* utility shall select only batch jobs for which the value of the *Priority* attribute of the batch job relates to the value of the *priority* component of the option-argument in the manner indicated by the value of the *op* component of the option-argument.

If the *op* component of the option-argument is omitted, the *qselect* utility shall select batch jobs for which the value of the *Priority* attribute of the batch job is equal to the value of the *priority* component of the option-argument.

When comparing priority values, the *qselect* utility shall use the following definitions for the *op* component of the option-argument:

`.eq.` The value of the *Priority* attribute of the batch job equals the value of the *priority* component of the option-argument.

`.ge.` The value of the *Priority* attribute of the batch job is greater than or equal to the value of the *priority* component option-argument.

`.gt.` The value of the *Priority* attribute of the batch job is greater than the value of the *priority* component option-argument.

`.lt.` The value of the *Priority* attribute of the batch job is less than the value of the *priority* component option-argument.

`.lt.` The value of the *Priority* attribute of the batch job is less than or equal to the value of the *priority* component option-argument.

`.ne.` The value of the *Priority* attribute of the batch job does not equal the value of the *priority* component option-argument.

–q *destination*

Restrict selection to the specified batch queue or server, or both.

The *qselect* utility shall select only batch jobs that are located at the destination indicated by the value of the *destination* option-argument.

The destination defines a batch queue, a server, or a batch queue at a server.

The *qselect* utility shall accept an option-argument for the **–q** option that conforms to the syntax for a destination. If the **–q** option is not presented to the *qselect* utility, the utility shall select batch jobs from all batch queues at the default batch server.

If the option-argument describes only a batch queue, the *qselect* utility shall select only batch jobs from the batch queue of the specified name at the default batch server. The means by which *qselect* determines the default server is implementation-defined.

If the option-argument describes only a batch server, the *qselect* utility shall select batch jobs from all the batch queues at that batch server.

If the option-argument describes both a batch queue and a batch server, the *qselect* utility shall select only batch jobs from the specified batch queue at the specified

server.

–r y | n — Restrict selection to batch jobs with the specified rerunability status.

The *qselect* utility shall select only batch jobs for which the value of the *Rerunable* attribute of the batch job matches the value of the option-argument.

The *qselect* utility shall accept a value for the option-argument that consists of either the single character 'y' or the single character 'n'. The character 'y' represents the value TRUE, and the character 'n' represents the value FALSE.

–s *states* — Restrict selection to batch jobs in the specified states.

The *qselect* utility shall accept an option-argument that consists of any combination of the characters 'e', 'q', 'r', 'w', 'h', and 't'.

Conforming applications shall not repeat any character in the option-argument. The *qselect* utility shall permit the repetition of characters in the option-argument, but shall not assign additional meaning to repeated characters.

The *qselect* utility shall interpret the characters in the *states* option-argument as follows:

e Represents the EXITING state.

q Represents the QUEUED state.

r Represents the RUNNING state.

t Represents the TRANSITING state.

h Represents the HELD state.

w Represents the WAITING state.

For each character in the *states* option-argument, the *qselect* utility shall select batch jobs in the corresponding state.

–u *user_list* — Restrict selection to batch jobs owned by the specified user names.

The *qselect* utility shall select only the batch jobs of those users specified in the *user_list* option-argument.

The *qselect* utility shall accept a *user_list* option-argument that conforms to the following syntax:

username[@*host*][,,*username*[@*host*],, ...]

The *qselect* utility shall accept only one user name that is missing a corresponding host name. The *qselect* utility shall accept only one user name per named host.

OPERANDS

None.

STDIN

Not used.

INPUT FILES

None.

30576 **ENVIRONMENT VARIABLES**

30577 The following environment variables shall affect the execution of *qselect*:

30578 *LANG* Provide a default value for the internationalization variables that are unset or null.
30579 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
30580 Internationalization Variables for the precedence of internationalization variables
30581 used to determine the values of locale categories.)

30582 *LC_ALL* If set to a non-empty string value, override the values of all the other
30583 internationalization variables.

30584 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
30585 characters (for example, single-byte as opposed to multi-byte characters in
30586 arguments).

30587 *LC_MESSAGES*
30588 Determine the locale that should be used to affect the format and contents of
30589 diagnostic messages written to standard error.

30590 *LOGNAME* Determine the login name of the user.

30591 *TZ* Determine the timezone used to interpret the *date-time* option-argument. If *TZ* is
30592 unset or null, an unspecified default timezone shall be used.

30593 **ASYNCHRONOUS EVENTS**

30594 Default.

30595 **STDOUT**

30596 The *qselect* utility shall write zero or more batch *job_identifier*s to standard output.

30597 The *qselect* utility shall separate the batch *job_identifier*s written to standard output by white
30598 space.

30599 The *qselect* utility shall write batch *job_identifier*s in the following format:

30600 `sequence_number.server_name@server`

30601 **STDERR**

30602 The standard error shall be used only for diagnostic messages.

30603 **OUTPUT FILES**

30604 None.

30605 **EXTENDED DESCRIPTION**

30606 None.

30607 **EXIT STATUS**

30608 The following exit values shall be returned:

30609 0 Successful completion.

30610 >0 An error occurred.

30611 **CONSEQUENCES OF ERRORS**

30612 Default.

APPLICATION USAGE

None.

EXAMPLES

The following example shows how a user might use the *qselect* utility in conjunction with the *qdel* utility to delete all of his or her jobs in the queued state without affecting any jobs that are already running:

```
qdel $(qselect –s q)
```

or:

```
qselect –s q || xargs qdel
```

RATIONALE

The *qselect* utility allows users to acquire a list of job identifiers that match user-specified selection criteria. The list of identifiers returned by the *qselect* utility conforms to the syntax of the batch job identifier list processed by a utility such as *qmove*, *qdel*, and *qrls*.The *qselect* utility is thus a powerful tool for causing another batch system utility to act upon a set of jobs that match a list of selection criteria.

The options of the *qselect* utility let the user apply a number of useful filters for selecting jobs. Each option further restricts the selection of jobs. Many of the selection options allow the specification of a relational operator. The FORTRAN-like syntax of the operator—that is, `".lt."`, was chosen rather than the C-like `"<="` meta-characters.

The **–a** option allows users to restrict the selected jobs to those that have been submitted (or altered) to wait until a particular time. The time period is determined by the argument of this option, which includes both a time and an operator—it is thus possible to select jobs waiting until a specific time, jobs waiting until after a certain time, or those waiting for a time before the specified time.

The **–A** option allows users to restrict the selected jobs to those that have been submitted (or altered) to charge a particular account.

The **–c** option allows users to restrict the selected jobs to those whose checkpointing interval falls within the specified range.

The **–l** option allows users to select those jobs whose resource limits fall within the range indicated by the value of the option. For example, a user could select those jobs for which the CPU time limit is greater than two hours.

The **–N** option allows users to select jobs by job name. For instance, all the parts of a task that have been divided in parallel jobs might be given the same name, and thus manipulated as a group by means of this option.

The **–q** option allows users to select jobs in a specified queue.

The **–r** option allows users to select only those jobs with a specified rerun criteria. For instance, a user might select only those jobs that can be rerun for use with the *qrerun* utility.

The **–s** option allows users to select only those jobs that are in a certain state.

The **–u** option allows users to select jobs that have been submitted to execute under a particular account.

The selection criteria provided by the options of the *qselect* utility allow users to select jobs based on all the appropriate attributes that can be assigned to jobs by the *qsub* utility.

Historically, the *qselect* utility has not been a part of existing practice; it is an improvement that has been introduced in this volume of IEEE Std 1003.1-200x.

30657 **FUTURE DIRECTIONS**

30658 None.

30659 **SEE ALSO**

30660 *qdel*, *qrerun*, *qrls*, *qselect*, *qsub*, *touch*, Chapter 3 (on page 2303)

30661 **CHANGE HISTORY**

30662 Derived from IEEE Std 1003.2d-1994.

30663 **NAME**

30664 qsig — signal batch jobs

30665 **SYNOPSIS**

30666 BE
```
qsig [–s signal] job_identifier ...
```
30667

30668 **DESCRIPTION**

30669 To signal a batch job is to send a signal to the session leader of the batch job. A batch job is
30670 signaled by sending a request to the batch server that manages the batch job. The *qsig* utility is a
30671 user-accessible batch client that requests the signaling of a batch job.

30672 The *qsig* utility shall signal those batch jobs for which a batch *job_identifier* is presented to the
30673 utility. The *qsig* utility shall not signal any batch jobs whose batch *job_identifier*s are not
30674 presented to the utility.

30675 The *qsig* utility shall signal batch jobs in the order in which the corresponding batch
30676 *job_identifier*s are presented to the utility. If the *qsig* utility fails to process a batch *job_identifier*
30677 successfully, the utility shall proceed to process the remaining batch *job_identifier*s, if any.

30678 The *qsig* utility shall signal batch jobs by sending a *Signal Job Request* to the batch server that
30679 manages the batch job.

30680 For each successfully processed batch *job_identifier*, the *qsig* utility shall have received a
30681 completion reply to each *Signal Job Request* sent to a batch server at the time the utility exits.

30682 **OPTIONS**

30683 The *qsig* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
30684 12.2, Utility Syntax Guidelines.

30685 The following option shall be supported by the implementation:

30686 **–s** *signal* Define the signal to be sent to the batch job.

30687 The *qsig* utility shall accept a *signal* option-argument that is either a symbolic
30688 signal name or an unsigned integer signal number (see the POSIX.1-1990 standard,
30689 Section 3.3.1.1). The *qsig* utility shall accept signal names for which the SIG prefix
30690 has been omitted.

30691 If the *signal* option-argument is a signal name, the *qsig* utility shall send that name.

30692 If the *signal* option-argument is a number, the *qsig* utility shall send the signal
30693 value represented by the number.

30694 If the **–s** option is not presented to the *qsig* utility, the utility shall send the signal
30695 SIGTERM to each signaled batch job.

30696 **OPERANDS**

30697 The *qsig* utility shall accept one or more operands that conform to the syntax for a batch
30698 *job_identifier* (see Section 3.3.1 (on page 2324)).

30699 **STDIN**

30700 Not used.

30701 **INPUT FILES**

30702 None.

30703 **ENVIRONMENT VARIABLES**

30704 The following environment variables shall affect the execution of *qsig*:

30705 *LANG* Provide a default value for the internationalization variables that are unset or null.
30706 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
30707 Internationalization Variables for the precedence of internationalization variables
30708 used to determine the values of locale categories.)

30709 *LC_ALL* If set to a non-empty string value, override the values of all the other
30710 internationalization variables.

30711 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
30712 characters (for example, single-byte as opposed to multi-byte characters in
30713 arguments).

30714 *LC_MESSAGES*
30715 Determine the locale that should be used to affect the format and contents of
30716 diagnostic messages written to standard error.

30717 *LOGNAME* Determine the login name of the user.

30718 **ASYNCHRONOUS EVENTS**

30719 Default.

30720 **STDOUT**

30721 An implementation of the *qsig* utility may write informative messages to standard output.

30722 **STDERR**

30723 The standard error shall be used only for diagnostic messages. |

30724 **OUTPUT FILES**

30725 None.

30726 **EXTENDED DESCRIPTION**

30727 None.

30728 **EXIT STATUS**

30729 The following exit values shall be returned:

30730 0 Successful completion.

30731 >0 An error occurred.

30732 **CONSEQUENCES OF ERRORS**

30733 In addition to the default behavior, the *qsig* utility shall not be required to write a diagnostic
30734 message to standard error when the error reply received from a batch server indicates that the
30735 batch *job_identifier* does not exist on the server. Whether or not the *qsig* utility waits to output the
30736 diagnostic message while attempting to locate the batch job on other servers is implementation-
30737 defined.

30738 **APPLICATION USAGE**

30739 None.

30740 **EXAMPLES**

30741 None.

30742 **RATIONALE**

30743 The *qsig* utility allows users to signal batch jobs.

30744 A user may be unable to signal a batch job with the *kill* utility of the operating system for a
30745 number of reasons. First, the process ID of the batch job may be unknown to the user. Second,

the processes of the batch job may be on a remote node. However, by virtue of communication between batch nodes, the *qsig* utility can arrange for the signaling of a process.

Because a batch job that is not running cannot be signaled, and because the signal may not terminate the batch job, the *qsig* utility is not a substitute for the *qdel* utility.

The options of the *qsig* utility allow the user to specify the signal that is to be sent to the batch job.

The **–s** option allows users to specify a signal by name or by number, and thus override the default signal. The POSIX.1-1990 standard defines signals by both name and number.

The *qsig* utility is a new utility, *vis-a-vis* existing practice; it has been defined in this volume of IEEE Std 1003.1-200x in response to user-perceived shortcomings in existing practice.

FUTURE DIRECTIONS

None.

SEE ALSO

kill, *qdel*, Chapter 3 (on page 2303)

CHANGE HISTORY

Derived from IEEE Std 1003.2d-1994.

Issue 6

The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

NAME

qstat — show status of batch jobs

SYNOPSIS

BE

```
qstat [–f] job_identifier ...
qstat –Q [–f] destination ...
qstat –B [–f] server_name ...
```

DESCRIPTION

The status of a batch job, batch queue, or batch server is obtained by a request to the server. The *qstat* utility is a user-accessible batch client that requests the status of one or more batch jobs, batch queues, or servers, and writes the status information to standard output.

For each successfully processed batch *job_identifier*, the *qstat* utility shall display information about the corresponding batch job.

For each successfully processed destination, the *qstat* utility shall display information about the corresponding batch queue.

For each successfully processed server name, the *qstat* utility shall display information about the corresponding server.

The *qstat* utility shall acquire batch job status information by sending a *Job Status Request* to a batch server. The *qstat* utility shall acquire batch queue status information by sending a *Queue Status Request* to a batch server. The *qstat* utility shall acquire server status information by sending a *Server Status Request* to a batch server.

OPTIONS

The *qstat* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported by the implementation:

–f Specify that a full display is produced.

The minimum contents of a full display are specified in the STDOUT section.

Additional contents and format of a full display are implementation-defined.

–Q Specify that the operand is a destination.

The *qstat* utility shall display information about each batch queue at each destination identified as an operand.

–B Specify that the operand is a server name.

The *qstat* utility shall display information about each server identified as an operand.

OPERANDS

If the **–Q** option is presented to the *qstat* utility, the utility shall accept one or more operands that conform to the syntax for a destination (see Section 3.3.2 (on page 2325)).

If the **–B** option is presented to the *qstat* utility, the utility shall accept one or more *server_name* operands.

If neither the **–B** nor the **–Q** option is presented to the *qstat* utility, the utility shall accept one or more operands that conform to the syntax for a batch *job_identifier* (see Section 3.3.1 (on page 2324)).

30806 **STDIN**

30807 Not used.

30808 **INPUT FILES**

30809 None.

30810 **ENVIRONMENT VARIABLES**

30811 The following environment variables shall affect the execution of *qstat*:

30812 *HOME* Determine the pathname of the user's home directory.

30813 *LANG* Provide a default value for the internationalization variables that are unset or null.
30814 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
30815 Internationalization Variables for the precedence of internationalization variables
30816 used to determine the values of locale categories.)

30817 *LC_ALL* If set to a non-empty string value, override the values of all the other
30818 internationalization variables.

30819 *LC_COLLATE*
30820 Determine the locale for the behavior of ranges, equivalence classes and multi-
30821 character collating elements within regular expressions.

30822 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
30823 characters (for example, single-byte as opposed to multi-byte characters in
30824 arguments).

30825 *LC_MESSAGES*
30826 Determine the locale that should be used to affect the format and contents of
30827 diagnostic messages written to standard error.

30828 *LC_NUMERIC*
30829 Determine the locale for selecting the radix character used when writing floating-
30830 point formatted output.

30831 **ASYNCHRONOUS EVENTS**

30832 Default.

30833 **STDOUT**

30834 If an operand presented to the *qstat* utility is a batch *job_identifier* and the **–f** option is not
30835 specified, the *qstat* utility shall display the following items on a single line, in the stated order,
30836 with white space between each item, for each successfully processed operand:

30837 • The batch *job_identifier*

30838 • The batch job name

30839 • The *Job_Owner* attribute

30840 • The CPU time used by the batch job

30841 • The batch job state

30842 • The batch job location

30843 If an operand presented to the *qstat* utility is a batch *job_identifier* and the **–f** option is specified,
30844 the *qstat* utility shall display the following items for each success fully processed operand:

30845 • The batch *job_identifier*

30846 • The batch job name

- The *Job_Owner* attribute
- The execution user ID
- The CPU time used by the batch job
- The batch job state
- The batch job location
- Additional implementation-defined information, if any, about the batch job or batch queue

If an operand presented to the *qstat* utility is a destination, the **–Q** option is specified, and the **–f** option is not specified, the *qstat* utility shall display the following items on a single line, in the stated order, with white space between each item, for each successfully processed operand:

- The batch queue name
- The maximum number of batch jobs that shall be run in the batch queue concurrently
- The total number of batch jobs in the batch queue
- The status of the batch queue
- For each state, the number of batch jobs in that state in the batch queue and the name of the state
- The type of batch queue (execution or routing)

If the operands presented to the *qstat* utility are destinations, the **–Q** option is specified, and the **–f** option is specified, the *qstat* utility shall display the following items for each successfully processed operand:

- The batch queue name
- The maximum number of batch jobs that shall be run in the batch queue concurrently
- The total number of batch jobs in the batch queue
- The status of the batch queue
- For each state, the number of batch jobs in that state in the batch queue and the name of the state
- The type of batch queue (execution or routing)
- Additional implementation-defined information, if any, about the batch queue

If the operands presented to the *qstat* utility are batch server names, the **–B** option is specified, and the **–f** option is not specified, the *qstat* utility shall display the following items on a single line, in the stated order, with white space between each item, for each successfully processed operand:

- The batch server name
- The maximum number of batch jobs that shall be run in the batch queue concurrently
- The total number of batch jobs managed by the batch server
- The status of the batch server
- For each state, the number of batch jobs in that state and the name of the state

If the operands presented to the *qstat* utility are server names, the **–B** option is specified, and the **–f** option is specified, the *qstat* utility shall display the following items for each successfully processed operand:

- The server name
- The maximum number of batch jobs that shall be run in the batch queue concurrently
- The total number of batch jobs managed by the server
- The status of the server
- For each state, the number of batch jobs in that state and the name of the state
- Additional implementation-defined information, if any, about the server

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

In addition to the default behavior, the *qstat* utility shall not be required to write a diagnostic message to standard error when the error reply received from a batch server indicates that the batch *job_identifier* does not exist on the server. Whether or not the *qstat* utility waits to output the diagnostic message while attempting to locate the batch job on other servers is implementation-defined.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

The *qstat* utility allows users to display the status of jobs and listing the batch jobs in queues.

The operands of the *qstat* utility may be either job identifiers, queues (specified as destination identifiers), or batch server names. The **–Q** and **–B** options, or absence thereof, indicate the nature of the operands.

The other options of the *qstat* utility allow the user to control the amount of information displayed and the format in which it is displayed. Should a user wish to display the status of a set of jobs that match a selection criteria, the *qselect* utility may be used to acquire such a list.

The **–f** option allows users to request a ‘‘full’’ display in an implementation-defined format.

Historically, the *qstat* utility has been a part of the NQS and its derivatives, the existing practice on which it is based.

FUTURE DIRECTIONS

None.

30925 **SEE ALSO**

30926 *qselect*, Chapter 3 (on page 2303)

30927 **CHANGE HISTORY**

30928 Derived from IEEE Std 1003.2d-1994.

30929 **Issue 6**

30930 IEEE PASC Interpretation 1003.2 #191 is applied, removing the following ENVIRONMENT
30931 VARIABLES listed as affecting *qstat*: *COLUMNS*, *LINES*, *LOGNAME*, *TERM*, and *TZ*.

30932 The *LC_TIME* entry is also removed from the ENVIRONMENT VARIABLES section.

30933 **NAME**

30934 qsub — submit a script

30935 **SYNOPSIS**

30936 BE
```
qsub [–a date_time][–A account_string][–c interval]
    [–C directive_prefix][–e path_name][–h][–j join_list][–k keep_list]
    [–m mail_options][–M mail_list][–N name]
    [–o path_name][–p priority][–q destination][–r y|n]
    [–S path_name_list][–u user_list][–v variable_list][–V]
    [–z][script]
```
30942

30943 **DESCRIPTION**

30944 To submit a script is to create a batch job that executes the script. A script is submitted by a
30945 request to a batch server. The *qsub* utility is a user-accessible batch client that submits a script.

30946 Upon successful completion, the *qsub* utility shall have created a batch job that will execute the
30947 submitted script.

30948 The *qsub* utility shall submit a script by sending a *Queue Job Request* to a batch server.

30949 The *qsub* utility shall place the value of the following environment variables in the *Variable_List*
30950 attribute of the batch job: *HOME*, *LANG*, *LOGNAME*, *PATH*, *MAIL*, *SHELL*, and *TZ*. The name
30951 of the environment variable shall be the current name prefixed with the string PBS_O_.

30952 **Note:** If the current value of the *HOME* variable in the environment space of the *qsub* utility is
30953 **/aa/bb/cc**, then *qsub* shall place *PBS_O_HOME*=**/aa/bb/cc** in the *Variable_List* attribute of the
30954 batch job.

30955 In addition to the variables described above, the *qsub* utility shall add the following variables
30956 with the indicated values to the variable list:

30957 *PBS_O_WORKDIR* The absolute path of the current working directory of the *qsub* utility
30958 process.

30959 *PBS_O_HOST* The name of the host on which the *qsub* utility is running.

30960 **OPTIONS**

30961 The *qsub* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
30962 12.2, Utility Syntax Guidelines.

30963 The following options shall be supported by the implementation:

30964 **–a** *date_time* Define the time at which a batch job becomes eligible for execution.

30965 The *qsub* utility shall accept an option-argument that conforms to the syntax of the
30966 *time* operand of the *touch* utility.

Table 4-18 Environment Variable Values (Utilities)

Variable Name	Value at qsub Time
PBS_O_HOME	*HOME*
PBS_O_HOST	Client host name
PBS_O_LANG	*LANG*
PBS_O_LOGNAME	*LOGNAME*
PBS_O_PATH	*PATH*
PBS_O_MAIL	*MAIL*
PBS_O_SHELL	*SHELL*
PBS_O_TZ	*TZ*
PBS_O_WORKDIR	Current working directory

Note: The server that initiates execution of the batch job will add other variables to the batch job's environment; see Section 3.2.2.1 (on page 2308).

The *qsub* utility shall set the *Execution_Time* attribute of the batch job to the number of seconds since the Epoch that is equivalent to the local time expressed by the value of the *date_time* option-argument. The Epoch is defined in the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.149, Epoch.

If the **–a** option is not presented to the *qsub* utility, the utility shall set the *Execution_Time* attribute of the batch job to a time (number of seconds since the Epoch) that is earlier than the time at which the utility exits.

–A *account_string*

Define the account to which the resource consumption of the batch job should be charged.

The syntax of the *account_string* option-argument is unspecified.

The *qsub* utility shall set the *Account_Name* attribute of the batch job to the value of the *account_string* option-argument.

If the **–A** option is not presented to the *qsub* utility, the utility shall omit the *Account_Name* attribute from the attributes of the batch job.

–c *interval* Define whether the batch job should be checkpointed, and if so, how often.

The *qsub* utility shall accept a value for the interval option-argument that is one of the following:

n — No checkpointing shall be performed on the batch batch job (NO_CHECKPOINT).

s — Checkpointing shall be performed only when the batch server is shut down (CHECKPOINT_AT_SHUTDOWN).

c — Automatic periodic checkpointing shall be performed at the *Minimum_Cpu_Interval* attribute of the batch queue, in units of CPU minutes (CHECKPOINT_AT_MIN_CPU_INTERVAL).

c=*minutes* — Automatic periodic checkpointing shall be performed every minutes of CPU time, or every *Minimum_Cpu_Interval* minutes, whichever is greater. The *minutes* argument shall conform to the syntax for unsigned integers and shall be greater than zero.

The *qsub* utility shall set the *Checkpoint* attribute of the batch job to the value of the *interval* option-argument.

If the **–c** option is not presented to the *qsub* utility, the utility shall set the *Checkpoint* attribute of the batch job to the single character 'u' (CHECKPOINT_UNSPECIFIED).

–C *directive_prefix*
Define the prefix that declares a directive to the *qsub* utility within the script.

The *directive_prefix* is not a batch job attribute; it affects the behavior of the *qsub* utility.

If the **–C** option is presented to the *qsub* utility, and the value of the *directive_prefix* option-argument is the null string, the utility shall not scan the script file for directives. If the **–C** option is not presented to the *qsub* utility, then the value of the *PBS_DPREFIX* environment variable is used. If the environment variable is not defined, then #PBS encoded in the portable character set is the default.

–e *path_name* Define the path to be used for the standard error stream of the batch job.

The *qsub* utility shall accept a *path_name* option-argument which can be preceded by a host name element of the form *hostname*:.

If the *path_name* option-argument constitutes an absolute pathname, the *qsub* utility shall set the *Error_Path* attribute of the batch job to the value of the *path_name* option-argument.

If the *path_name* option-argument constitutes a relative pathname and no host name element is specified, the *qsub* utility shall set the *Error_Path* attribute of the batch job to the value of the absolute pathname derived by expanding the *path_name* option-argument relative to the current directory of the process executing *qsub*.

If the *path_name* option-argument constitutes a relative pathname and a host name element is specified, the *qsub* utility shall set the *Error_Path* attribute of the batch job to the value of the *path_name* option-argument without expansion. The host name element shall be included.

If the *path_name* option-argument does not include a host name element, the *qsub* utility shall prefix the pathname with *hostname*:, where *hostname* is the name of the host upon which the *qsub* utility is being executed.

If the **–e** option is not presented to the *qsub* utility, the utility shall set the *Error_Path* attribute of the batch job to the host name and path of the current directory of the submitting process and the default filename.

The default filename for standard error has the following format:

job_name.e*sequence_number*

–h Specify that a USER hold is applied to the batch job.

The *qsub* utility shall set the value of the *Hold_Types* attribute of the batch job to the value USER.

If the **–h** option is not presented to the *qsub* utility, the utility shall set the *Hold_Types* attribute of the batch job to the value NO_HOLD.

–j *join_list* Define which streams of the batch job are to be merged. The *qsub* **–j** option shall accept a value for the *join_list* option-argument that is a string of alphanumeric characters in the portable character set (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set).

The *qsub* utility shall accept a *join_list* option-argument that consists of one or more of the characters 'e' and 'o' or the single character 'n'.

All of the other batch job output streams specified will be merged into the output stream represented by the character listed first in the *join_list* option-argument.

For each unique character in the *join_list* option-argument, the *qsub* utility shall add a value to the *Join_Path* attribute of the batch job as follows, each representing a different batch job stream to join:

e The standard error of the batch batch job (JOIN_STD_ERROR).

o The standard output of the batch batch job (JOIN_STD_OUTPUT).

An existing *Join_Path* attribute can be cleared by the following join type:

n NO_JOIN

If 'n' is specified, then no files are joined. The *qsub* utility shall consider it an error if any join type other than 'n' is combined with join type 'n'.

Strictly conforming applications shall not repeat any of the characters 'e', 'o', or 'n' within the *join_list* option-argument. The *qsub* utility shall permit the repetition of characters, but shall not assign additional meaning to the repeated characters.

An implementation may define other join types. The conformance document for an implementation shall describe any additional batch job streams, how they are specified, their internal behavior, and how they affect the behavior of the utility.

If the **–j** option is not presented to the *qsub* utility, the utility shall set the value of the *Join_Path* attribute of the batch job to NO_JOIN.

–k *keep_list* Define which output of the batch job to retain on the execution host.

The *qsub* **–k** option shall accept a value for the *keep_list* option-argument that is a string of alphanumeric characters in the portable character set (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set).

The *qsub* utility shall accept a *keep_list* option-argument that consists of one or more of the characters 'e' and 'o' or the single character 'n'.

For each unique character in the *keep_list* option-argument, the *qsub* utility shall add a value to the *Keep_Files* attribute of the batch job as follows, each representing a different batch job stream to keep:

e The standard error of the batch batch job (KEEP_STD_ERROR).

o The standard output of the batch batch job (KEEP_STD_OUTPUT).

If both 'e' and 'o' are specified, then both files are retained. An existing *Keep_Files* attribute can be cleared by the following keep type:

n NO_KEEP

If 'n' is specified, then no files are retained. The *qsub* utility shall consider it an error if any keep type other than 'n' is combined with keep type 'n'.

Strictly conforming applications shall not repeat any of the characters 'e', 'o', or 'n' within the *keep_list* option-argument. The *qsub* utility shall permit the repetition of characters, but shall not assign additional meaning to the repeated characters.

An implementation may define other keep types. The conformance document for an implementation shall describe any additional keep types, how they are specified, their internal behavior, and how they affect the behavior of the utility. If the **–k** option is not presented to the *qsub* utility, the utility shall set the *Keep_Files* attribute of the batch job to the value NO_KEEP.

–m *mail_options*

Define the points in the execution of the batch job at which the batch server that manages the batch job shall send mail about a change in the state of the batch job.

The *qsub* **–m** option shall accept a value for the *mail_options* option-argument that is a string of alphanumeric characters in the portable character set (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character Set).

The *qsub* utility shall accept a value for the *mail_options* option-argument that is a string of one or more of the characters `'e'`, `'b'`, and `'a'`, or the single character `'n'`.

For each unique character in the *mail_options* option-argument, the *qsub* utility shall add a value to the *Mail_Users* attribute of the batch job as follows, each representing a different time during the life of a batch job at which to send mail:

e MAIL_AT_EXIT

b MAIL_AT_BEGINNING

a MAIL_AT_ABORT

If any of these characters are duplicated in the *mail_options* option-argument, the duplicates shall be ignored.

An existing *Mail_Points* attribute can be cleared by the following mail type:

n NO_MAIL

If `'n'` is specified, then mail is not sent. The *qsub* utility shall consider it an error if any mail type other than `'n'` is combined with mail type `'n'`.

Strictly conforming applications shall not repeat any of the characters `'e'`, `'b'`, `'a'`, or `'n'` within the *mail_options* option-argument.

The *qsub* utility shall permit the repetition of characters, but shall not assign additional meaning to the repeated characters. An implementation may define other mail types. The conformance document for an implementation shall describe any additional mail types, how they are specified, their internal behavior, and how they affect the behavior of the utility.

If the **–m** option is not presented to the *qsub* utility, the utility shall set the *Mail_Points* attribute to the value MAIL_AT_ABORT.

–M *mail_list* Define the list of users to which a batch server that executes the batch job shall send mail, if the server sends mail about the batch job.

The syntax of the *mail_list* option-argument is unspecified.

If the implementation of the *qsub* utility uses a name service to locate users, the utility should accept the syntax used by the name service.

If the implementation of the *qsub* utility does not use a name service to locate users, the implementation should accept the following syntax for user names:

31139 *mail_address*[,,*mail_address*,, ...]

31140 The interpretation of *mail_address* is implementation-defined.

31141 The *qsub* utility shall set the *Mail_Users* attribute of the batch job to the value of the
31142 *mail_list* option-argument.

31143 If the –**M** option is not presented to the *qsub* utility, the utility shall place only the
31144 user name and host name for the current process in the *Mail_Users* attribute of the
31145 batch job.

31146 –**N** *name* Define the name of the batch job.

31147 The *qsub* –**N** option shall accept a value for the *name* option-argument that is a
31148 string of up to 15 alphanumeric characters in the portable character set (see the
31149 Base Definitions volume of IEEE Std 1003.1-200x, Section 6.1, Portable Character
31150 Set) where the first character is alphabetic.

31151 The *qsub* utility shall set the value of the *Job_Name* attribute of the batch job to the
31152 value of the *name* option-argument.

31153 If the –**N** option is not presented to the *qsub* utility, the utility shall set the
31154 *Job_Name* attribute of the batch job to the name of the *script* argument from which
31155 the directory specification if any, has been removed.

31156 If the –**N** option is not presented to the *qsub* utility, and the script is read from
31157 standard input, the utility shall set the *Job_Name* attribute of the batch job to the
31158 value STDIN.

31159 –**o** *path_name* Define the path for the standard output of the batch job.

31160 The *qsub* utility shall accept a *path_name* option-argument that conforms to the
31161 syntax of the *path_name* element defined in the System Interfaces volume of
31162 IEEE Std 1003.1-200x, which can be preceded by a host name element of the form
31163 *hostname*:.

31164 If the *path_name* option-argument constitutes an absolute pathname, the *qsub*
31165 utility shall set the *Output_Path* attribute of the batch job to the value of the
31166 *path_name* option-argument without expansion.

31167 If the *path_name* option-argument constitutes a relative pathname and no host
31168 name element is specified, the *qsub* utility shall set the *Output_Path* attribute of the
31169 batch job to the pathname derived by expanding the value of the *path_name*
31170 option-argument relative to the current directory of the process executing the *qsub*.

31171 If the *path_name* option-argument constitutes a relative pathname and a host name
31172 element is specified, the *qsub* utility shall set the *Output_Path* attribute of the batch
31173 job to the value of the *path_name* option-argument without expansion.

31174 If the *path_name* option-argument does not specify a host name element, the *qsub*
31175 utility shall prefix the pathname with *hostname*:, where *hostname* is the name of the
31176 host upon which the *qsub* utility is executing.

31177 If the –**o** option is not presented to the *qsub* utility, the utility shall set the
31178 *Output_Path* attribute of the batch job to the host name and path of the current
31179 directory of the submitting process and the default filename.

31180 The default filename for standard output has the following format:

31181 *job_name*.o*sequence_number*

31182 –**p** *priority* Define the priority the batch job should have relative to other batch jobs owned by
31183 the batch server.

31184 The *qsub* utility shall set the *Priority* attribute of the batch job to the value of the
31185 *priority* option-argument.

31186 If the –**p** option is not presented to the *qsub* utility, the value of the *Priority*
31187 attribute is implementation-defined.

31188 The *qsub* utility shall accept a value for the *priority* option-argument that conforms
31189 to the syntax for signed decimal integers, and which is not less than –1 024 and not
31190 greater than 1 023.

31191 –**q** *destination*
31192 Define the destination of the batch job.

31193 The destination is not a batch job attribute; it determines the batch server, and
31194 possibly the batch queue, to which the *qsub* utility batch queues the batch job.

31195 The *qsub* utility shall submit the script to the batch server named by the *destination*
31196 option-argument or the server that owns the batch queue named in the *destination*
31197 option-argument.

31198 The *qsub* utility shall accept an option-argument for the –**q** option that conforms to
31199 the syntax for a destination (see Section 3.3.2 (on page 2325)).

31200 If the –**q** option is not presented to the *qsub* utility, the *qsub* utility shall submit the
31201 batch job to the default destination. The mechanism for determining the default
31202 destination is implementation-defined.

31203 –**r** *y* | *n* Define whether the batch job is rerunable.

31204 If the value of the option-argument is *y*, the *qsub* utility shall set the *Rerunable*
31205 attribute of the batch job to TRUE.

31206 If the value of the option-argument is *n*, the *qsub* utility shall set the *Rerunable*
31207 attribute of the batch job to FALSE.

31208 If the –**r** option is not presented to the *qsub* utility, the utility shall set the *Rerunable*
31209 attribute of the batch job to TRUE.

31210 –**S** *path_name_list*
31211 Define the pathname to the shell under which the batch job is to execute.

31212 The *qsub* utility shall accept a *path_name_list* option-argument that conforms to the
31213 following syntax:

31214 ***pathname*[@*host*][,,*pathname*[@*host*],, ...]**

31215 The *qsub* utility shall allow only one pathname for a given host name. The *qsub*
31216 utility shall allow only one pathname that is missing a corresponding host name.

31217 The *qsub* utility shall add a value to the *Shell_Path_List* attribute of the batch job for
31218 each entry in the *path_name_list* option-argument.

31219 If the –**S** option is not presented to the *qsub* utility, the utility shall set the
31220 *Shell_Path_List* attribute of the batch job to the null string.

31221 The conformance document for an implementation shall describe the mechanism
31222 used to set the default shell and determine the current value of the default shell.
31223 An implementation shall provide a means for the installation to set the default
31224 shell to the login shell of the user under which the batch job is to execute. See

Section 3.3.3 (on page 2325) for a means of removing *keyword=value* (and *value@keyword*) pairs and other general rules for list-oriented batch job attributes.

–u *user_list* Define the user name under which the batch job is to execute.

The *qsub* utility shall accept a *user_list* option-argument that conforms to the following syntax:

```
username[@host][,,username[@host],, ...]
```

The *qsub* utility shall accept only one user name that is missing a corresponding host name. The *qsub* utility shall accept only one user name per named host.

The *qsub* utility shall add a value to the *User_List* attribute of the batch job for each entry in the *user_list* option-argument.

If the **–u** option is not presented to the *qsub* utility, the utility shall set the *User_List* attribute of the batch job to the user name from which the utility is executing. See Section 3.3.3 (on page 2325) for a means of removing *keyword=value* (and *value@keyword*) pairs and other general rules for list-oriented batch job attributes.

–v *variable_list*
Add to the list of variables that are exported to the session leader of the batch job.

A *variable_list* is a set of strings of either the form <*variable*> or <*variable=value*>, delimited by commas.

If the **–v** option is presented to the *qsub* utility, the utility shall also add, to the environment *Variable_List* attribute of the batch job, every variable named in the environment *variable_list* option-argument and, optionally, values of specified variables.

If a value is not provided on the command line, the *qsub* utility shall set the value of each variable in the environment *Variable_List* attribute of the batch job to the value of the corresponding environment variable for the process in which the utility is executing; see Table 4-18 (on page 3003).

A conforming application shall not repeat a variable in the environment *variable_list* option-argument.

The *qsub* utility shall not repeat a variable in the environment *Variable_List* attribute of the batch job. See Section 3.3.3 (on page 2325) for a means of removing *keyword=value* (and *value@keyword*) pairs and other general rules for list-oriented batch job attributes.

–V Specify that all of the environment variables of the process are exported to the context of the batch job.

The *qsub* utility shall place every environment variable in the process in which the utility is executing in the list and shall set the value of each variable in the attribute to the value of that variable in the process.

–z Specify that the utility does not write the batch *job_identifier* of the created batch job to standard output.

If the **–z** option is presented to the *qsub* utility, the utility shall not write the batch *job_identifier* of the created batch job to standard output.

If the **–z** option is not presented to the *qsub* utility, the utility shall write the identifier of the created batch job to standard output.

31268 **OPERANDS**

31269 The *qsub* utility shall accept a *script* operand that indicates the path to the script of the batch job.

31270 If the *script* operand is not presented to the *qsub* utility, or if the operand is the single-character
31271 string `'-'`, the utility shall read the script from standard input.

31272 If the script represents a partial path, the *qsub* utility shall expand the path relative to the current
31273 directory of the process executing the utility.

31274 **STDIN**

31275 The *qsub* utility reads the script of the batch job from standard input if the script operand is
31276 omitted or is the single character `'-'`.

31277 **INPUT FILES**

31278 In addition to binding the file indicated by the *script* operand to the batch job, the *qsub* utility
31279 reads the script file and acts on directives in the script.

31280 **ENVIRONMENT VARIABLES**

31281 The following environment variables shall affect the execution of *qsub*:

31282 *LANG* Provide a default value for the internationalization variables that are unset or null.
31283 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
31284 Internationalization Variables for the precedence of internationalization variables
31285 used to determine the values of locale categories.)

31286 *LC_ALL* If set to a non-empty string value, override the values of all the other
31287 internationalization variables.

31288 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
31289 characters (for example, single-byte as opposed to multi-byte characters in
31290 arguments).

31291 *LC_MESSAGES*
31292 Determine the locale that should be used to affect the format and contents of
31293 diagnostic messages written to standard error.

31294 *LOGNAME* Determine the login name of the user.

31295 *PBS_DPREFIX*
31296 Determine the default prefix for directives within the script.

31297 *SHELL* Determine the pathname of the preferred command language interpreter of the
31298 user.

31299 *TZ* Determine the timezone used to interpret the *date-time* option-argument. If *TZ* is
31300 unset or null, an unspecified default timezone shall be used.

31301 **ASYNCHRONOUS EVENTS**

31302 Once created, a batch job exists until it exits, aborts, or is deleted.

31303 After a batch job is created by the *qsub* utility, batch servers might route, execute, modify, or
31304 delete the batch job.

31305 **STDOUT**

31306 The *qsub* utility writes the batch *job_identifier* assigned to the batch job to standard output, unless
31307 the **–z** option is specified.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

Script Preservation

The *qsub* utility shall make the script available to the server executing the batch job in such a way that the server executes the script as it exists at the time of submission.

The *qsub* utility can send a copy of the script to the server with the *Queue Job Request* or store a temporary copy of the script in a location specified to the server.

Option Specification

A script can contain directives to the *qsub* utility.

The *qsub* utility shall scan the lines of the script for directives, skipping blank lines, until the first line that begins with a string other than the directive string; if directives occur on subsequent lines, the utility shall ignore those directives.

Lines are separated by a <newline>. If the first line of the script begins with "#!" or a colon (':'), then it is skipped. The *qsub* utility shall process a line in the script as a directive if and only if the string of characters from the first non-white-space character on the line until the first <space> or <tab> on the line match the directive prefix. If a line in the script contains a directive and the final characters of the line are backslash ('\') and <newline>, then the next line shall be interpreted as a continuation of that directive.

The *qsub* utility shall process the options and option-arguments contained on the directive prefix line using the same syntax as if the options were input on the *qsub* utility.

The *qsub* utility shall continue to process a directive prefix line until after a <newline> is encountered. An implementation may ignore lines which, according to the syntax of the shell that will interpret the script, are comments. An implementation shall describe in the conformance document the format of any shell comments that it will recognize.

If an option is present in both a directive and the arguments to the *qsub* utility, the utility shall ignore the option and the corresponding option-argument, if any, in the directive.

If an option that is present in the directive is not present in the arguments to the *qsub* utility, the utility shall process the option and the option-argument, if any.

In order of preference, the *qsub* utility shall select the directive prefix from one of the following sources:

- If the **–C** option is presented to the utility, the value of the *directive_prefix* option-argument
- If the environment variable *PBS_DPREFIX* is defined, the value of that variable
- The four-character string "#PBS" encoded in the portable character set

If the **–C** option is present in the script file it shall be ignored.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

The *qsub* utility allows users to create a batch job that will process the script specified as the operand of the utility.

The options of the *qsub* utility allow users to control many aspects of the queuing and execution of a batch job.

The **–a** option allows users to designate the time after which the batch job will become eligible to run. By specifying an execution time, users can take advantage of resources at off-peak hours, synchronize jobs with chronologically predictable events, and perhaps take advantage of off-peak pricing of computing time. For these reasons and others, a timing option is existing practice on the part of almost every batch system, including NQS.

The **–A** option allows users to specify the account that will be charged for the batch job. Support for account is not mandatory for conforming batch servers.

The **–C** option allows users to prescribe the prefix for directives within the script file. The default prefix "#PBS" may be inappropriate if the script will be interpreted with an alternate shell, as specified by the **–S** option.

The **–c** option allows users to establish the checkpointing interval for their jobs. A checkpointing system, which is not defined by this volume of IEEE Std 1003.1-200x, allows recovery of a batch job at the most recent checkpoint in the event of a crash. Checkpointing is typically used for jobs that consume expensive computing time or must meet a critical schedule. Users should be allowed to make the tradeoff between the overhead of checkpointing and the risk to the timely completion of the batch job; therefore, this volume of IEEE Std 1003.1-200x provides the checkpointing interval option. Support for checkpointing is optional for batch servers.

The **–e** option allows users to redirect the standard error streams of their jobs to a non-default path. For example, if the submitted script generally produces a great deal of useless error output, a user might redirect the standard error output to the null device. Or, if the file system holding the default location (the home directory of the user) has too little free space, the user might redirect the standard error stream to a file in another file system.

The **–h** option allows users to create a batch job that is held until explicitly released. The ability to create a held job is useful when some external event must complete before the batch job can execute. For example, the user might submit a held job and release it when the system load has dropped.

The **–j** option allows users to merge the standard error of a batch job into its standard output stream, which has the advantage of showing the sequential relationship between output and error messages.

The **–m** option allows users to designate those points in the execution of a batch job at which mail will be sent to the submitting user, or to the account(s) indicated by the **–M** option. By requesting mail notification at points of interest in the life of a job, the submitting user, or other designated users, can track the progress of a batch job.

The **–N** option allows users to associate a name with the batch job. The job name in no way affects the processing of the batch job, but rather serves as a mnemonic handle for users. For example, the batch job name can help the user distinguish between multiple jobs listed by the *qstat* utility.

The **–o** option allows users to redirect the standard output stream. A user might, for example, wish to redirect to the null device the standard output stream of a job that produces copious yet superfluous output.

The **–P** option allows users to designate the relative priority of a batch job for selection from a queue.

The **–q** option allows users to specify an initial queue for the batch job. If the user specifies a routing queue, the batch batch server routes the batch job to another queue for execution or further routing. If the user specifies a non-routing queue, the batch server of the queue eventually executes the batch job.

The **–r** option allows users to control whether the submitted job will be rerun if the controlling batch node fails during execution of the batch job. The **–r** option likewise allows users to indicate whether or not the batch job is eligible to be rerun by the *qrerun* utility. Some jobs cannot be correctly rerun because of changes they make in the state of databases or other aspects of their environment. This volume of IEEE Std 1003.1-200x specifies that the default, if the **–r** option is not presented to the utility, will be that the batch job cannot be rerun, since the result of rerunning a non-rerunable job might be catastrophic.

The **–S** option allows users to specify the program (usually a shell) that will be invoked to process the script of the batch job. This option has been modified to allow a list of shell names and locations associated with different hosts.

The **–u** option is useful when the submitting user is authorized to use more than one account on a given host, in which case the **–u** option allows the user to select from among those accounts. The option-argument is a list of user-host pairs, so that the submitting user can provide different user identifiers for different nodes in the event the batch job is routed. The **–u** option provides a lot of flexibility to accommodate sites with complex account structures. Users that have the same user identifier on all the hosts they are authorized to use will not need to use the **–u** option.

The **–V** option allows users to export all their current environment variables, as of the time the batch job is submitted, to the context of the processes of the batch job.

The **–v** option allows users to export specific environment variables from their current process to the processes of the batch job.

The **–z** option allows users to suppress the writing of the batch job identifier to standard output. The **–z** option is an existing NQS practice that has been standardized.

Historically, the *qsub* utility has served the batch job-submission function in the NQS system, the existing practice on which it is based. Some changes and additions have been made to the *qsub* utility in this volume of IEEE Std 1003.1-200x, *vis-a-vis* NQS, as a result of the growing pool of experience with distributed batch systems.

The set of features of the *qsub* utility as defined in this volume of IEEE Std 1003.1-200x appears to incorporate all the common existing practice on potentially POSIX-conformant platforms.

FUTURE DIRECTIONS

None.

31436 **SEE ALSO**

31437 *qrerun*, *qstat*, *touch*, Chapter 3 (on page 2303)

31438 **CHANGE HISTORY**

31439 Derived from IEEE Std 1003.2d-1994.

31440 **Issue 6**

31441 The **–l** option has been removed as there is no portable description of the resources that are
31442 allowed or required by the batch job.

NAME

read — read a line from standard input

SYNOPSIS

```
read [–r] var...
```

DESCRIPTION

The *read* utility shall read a single line from standard input.

By default, unless the **–r** option is specified, backslash (`'\'`) shall act as an escape character, as described in Section 2.2.1 (on page 2232). 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.

The line shall be split into fields as in the shell (see Section 2.6.5 (on page 2243)); the first field shall be assigned to the first variable *var*, the second field to the second variable *var*, and so on. If there are fewer *var* operands specified than there are fields, the leftover fields and their intervening separators shall be assigned to the last *var*. If there are fewer fields than *var*s, the remaining *var*s shall be set to empty strings.

The setting of variables specified by the *var* operands shall affect the current shell execution environment; see Section 2.12 (on page 2263). If it is called in a subshell or separate utility execution environment, such as one of the following:

```
(read foo)
nohup read ...
find . –exec read ... \;
```

it shall not affect the shell variables in the caller's environment.

OPTIONS

The *read* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following option is supported:

–r Do not treat a backslash character in any special way. Consider each backslash to be part of the input line.

OPERANDS

The following operand shall be supported:

var The name of an existing or nonexisting shell variable.

STDIN

The standard input shall be a text file.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *read*:

IFS Determine the internal field separators used to delimit fields; see Section 2.5.3 (on page 2236).

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables

used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

PS2 Provide 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.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used for diagnostic messages and prompts for continued input.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 End-of-file was detected or an error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The **–r** option is included to enable *read* to subsume the purpose of the *line* utility, which is not included in IEEE Std 1003.1-200x.

The results are undefined if an end-of-file is detected following a backslash at the end of a line when **–r** is not specified.

EXAMPLES

The following command:

```
while read –r xx yy
do
    printf "%s %s\n" "$yy" "$xx"
done < input_file
```

prints a file with the first field of each line moved to the end of the line.

RATIONALE

The *read* utility historically has been a shell built-in. It was separated off into its own utility to take advantage of the richer description of functionality introduced by this volume of IEEE Std 1003.1-200x.

Since *read* affects the current shell execution environment, it is generally provided as a shell regular built-in. If it is called in a subshell or separate utility execution environment, such as one of the following:

```
(read foo)
nohup read ...
find . –exec read ... \;
```

it does not affect the shell variables in the environment of the caller.

FUTURE DIRECTIONS

None.

SEE ALSO

None.

CHANGE HISTORY

First released in Issue 2.

31544 **NAME**

31545 renice — set nice values of running processes

31546 **SYNOPSIS**

31547 UP
```
renice –n increment [–g | –p | –u] ID ...
```
31548

31549 **DESCRIPTION**

31550 The *renice* utility shall request that the nice values (see the Base Definitions volume of
31551 IEEE Std 1003.1-200x, Section 3.239, Nice Value) of one or more running processes be changed.
31552 By default, the applicable processes are specified by their process IDs. When a process group is
31553 specified (see **–g**), the request shall apply to all processes in the process group.

31554 The nice value shall be bounded in an implementation-defined manner. If the requested
31555 *increment* would raise or lower the nice value of the executed utility beyond implementation-
31556 defined limits, then the limit whose value was exceeded shall be used.

31557 When a user is *renice*d, the request applies to all processes whose saved set-user-ID matches the
31558 user ID corresponding to the user.

31559 Regardless of which options are supplied or any other factor, *renice* shall not alter the nice values
31560 of any process unless the user requesting such a change has appropriate privileges to do so for
31561 the specified process. If the user lacks appropriate privileges to perform the requested action, the
31562 utility shall return an error status.

31563 The saved set-user-ID of the user's process shall be checked instead of its effective user ID when
31564 *renice* attempts to determine the user ID of the process in order to determine whether the user
31565 has appropriate privileges.

31566 **OPTIONS**

31567 The *renice* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
31568 12.2, Utility Syntax Guidelines.

31569 The following options shall be supported:

31570 **–g** Interpret all operands as unsigned decimal integer process group IDs.

31571 **–n** *increment* Specify how the nice value of the specified process or processes is to be adjusted.
31572 The *increment* option-argument is a positive or negative decimal integer that shall
31573 be used to modify the nice value of the specified process or processes.

31574 Positive *increment* values shall cause a lower nice value. Negative *increment* values
31575 may require appropriate privileges and shall cause a higher nice value.

31576 **–p** Interpret all operands as unsigned decimal integer process IDs. The **–p** option is
31577 the default if no options are specified.

31578 **–u** Interpret all operands as users. If a user exists with a user name equal to the
31579 operand, then the user ID of that user is used in further processing. Otherwise, if
31580 the operand represents an unsigned decimal integer, it shall be used as the numeric
31581 user ID of the user.

31582 **OPERANDS**

31583 The following operands shall be supported:

31584 *ID* A process ID, process group ID, or user name/user ID, depending on the option
31585 selected.

31586 **STDIN**

31587 Not used.

31588 **INPUT FILES**

31589 None.

31590 **ENVIRONMENT VARIABLES**

31591 The following environment variables shall affect the execution of *renice*:

31592 *LANG* Provide a default value for the internationalization variables that are unset or null.
31593 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
31594 Internationalization Variables for the precedence of internationalization variables
31595 used to determine the values of locale categories.)

31596 *LC_ALL* If set to a non-empty string value, override the values of all the other
31597 internationalization variables.

31598 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
31599 characters (for example, single-byte as opposed to multi-byte characters in
31600 arguments).

31601 *LC_MESSAGES*
31602 Determine the locale that should be used to affect the format and contents of
31603 diagnostic messages written to standard error.

31604 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

31605 **ASYNCHRONOUS EVENTS**

31606 Default.

31607 **STDOUT**

31608 Not used.

31609 **STDERR**

31610 The standard error shall be used only for diagnostic messages.

31611 **OUTPUT FILES**

31612 None.

31613 **EXTENDED DESCRIPTION**

31614 None.

31615 **EXIT STATUS**

31616 The following exit values shall be returned:

31617 0 Successful completion.

31618 >0 An error occurred.

31619 **CONSEQUENCES OF ERRORS**

31620 Default.

APPLICATION USAGE

None.

EXAMPLES

1. Adjust the nice value so that process IDs 987 and 32 would have a lower nice value:

   ```
   renice –n 5 –p 987 32
   ```

2. Adjust the nice value so that group IDs 324 and 76 would have a higher nice value, if the user has the appropriate privileges to do so:

   ```
   renice –n –4 –g 324 76
   ```

3. Adjust the nice value so that numeric user ID 8 and user **sas** would have a lower nice value:

   ```
   renice –n 4 –u 8 sas
   ```

Useful nice value increments on historical systems include 19 or 20 (the affected processes run only when nothing else in the system attempts to run) and any negative number (to make processes run faster).

RATIONALE

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 section, rather than the OPERANDS section.

The definition of nice value is not intended to suggest that all processes in a system have priorities that are comparable. Scheduling policy extensions such as the realtime priorities in the System Interfaces volume of IEEE Std 1003.1-200x make the notion of a single underlying priority for all scheduling policies problematic. Some implementations may implement the *nice*-related features to affect all processes on the system, others to affect just the general time-sharing activities implied by this volume of IEEE Std 1003.1-200x, and others may have no effect at all. Because of the use of ‘‘implementation-defined’’ in *nice* and *renice*, a wide range of implementation strategies are possible.

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 a newer version of *renice*, which used the term ‘‘system scheduling priority’’, it was hoped that novice users could better understand what this utility was meant to do. Also, it would be easier to document what the utility was meant to do. Unfortunately, the addition of the POSIX realtime scheduling capabilities introduced the concepts of process and thread scheduling priorities that were totally unaffected by the *nice*/*renice* utilities or the *nice*()/*setpriority*() functions. Continuing to use the term ‘‘system scheduling priority’’ would have incorrectly suggested that these utilities and functions were indeed affecting these realtime priorities. It was decided to revert to the historical term ‘‘nice value’’ to reference this unrelated process attribute.

Although this utility has use by system administrators (and in fact appears in the system administration portion of the BSD documentation), the standard developers considered that it was very useful for individual end users to control their own processes.

FUTURE DIRECTIONS

None.

SEE ALSO

nice

CHANGE HISTORY

First released in Issue 4.

Issue 5

In the SYNOPSIS, an ellipsis is added to the **–u** option in all three obsolescent forms.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The APPLICATION USAGE section is added.

The obsolescent forms of the SYNOPSIS are removed.

Text previously conditional on POSIX_SAVED_IDS is mandatory in this issue. This is a FIPS requirement.

31675 **NAME**

31676 rm — remove directory entries

31677 **SYNOPSIS**

31678 `rm [–fiRr] file...`

31679 **DESCRIPTION**

31680 The *rm* utility shall remove the directory entry specified by each *file* argument.

31681 If either of the files dot or dot-dot are specified as the basename portion of an operand (that is,
31682 the final pathname component), *rm* shall write a diagnostic message to standard error and do
31683 nothing more with such operands.

31684 For each *file* the following steps shall be taken:

31685 1. If the *file* does not exist:

31686 a. If the **–f** option is not specified, *rm* shall write a diagnostic message to standard error. |

31687 b. Go on to any remaining *files*.

31688 2. If *file* is of type directory, the following steps shall be taken:

31689 a. If neither the **–R** option nor the **–r** option is specified, *rm* shall write a diagnostic |
31690 message to standard error, do nothing more with *file*, and go on to any remaining |
31691 files.

31692 b. If the **–f** option is not specified, and either the permissions of *file* do not permit
31693 writing and the standard input is a terminal or the **–i** option is specified, *rm* shall |
31694 write a prompt to standard error and read a line from the standard input. If the |
31695 response is not affirmative, *rm* shall do nothing more with the current file and go on |
31696 to any remaining files. |

31697 c. For each entry contained in *file*, other than dot or dot-dot, the four steps listed here (1 |
31698 to 4) shall be taken with the entry as if it were a *file* operand. The *rm* utility shall not |
31699 traverse directories by following symbolic links into other parts of the hierarchy, but
31700 shall remove the links themselves.

31701 d. If the **–i** option is specified, *rm* shall write a prompt to standard error and read a line |
31702 from the standard input. If the response is not affirmative, *rm* shall do nothing more |
31703 with the current file, and go on to any remaining files. |

31704 3. If *file* is not of type directory, the **–f** option is not specified, and either the permissions of
31705 *file* do not permit writing and the standard input is a terminal or the **–i** option is specified, |
31706 *rm* shall write a prompt to the standard error and read a line from the standard input. If the |
31707 response is not affirmative, *rm* shall do nothing more with the current file and go on to any |
31708 remaining files. |

31709 4. If the current file is a directory, *rm* shall perform actions equivalent to the *rmdir*( ) function
31710 defined in the System Interfaces volume of IEEE Std 1003.1-200x called with a pathname of
31711 the current file used as the *path* argument. If the current file is not a directory, *rm* shall
31712 perform actions equivalent to the *unlink*( ) function defined in the System Interfaces
31713 volume of IEEE Std 1003.1-200x called with a pathname of the current file used as the *path*
31714 argument.

31715 If this fails for any reason, *rm* shall write a diagnostic message to standard error, do
31716 nothing more with the current file, and go on to any remaining files.

31717 The *rm* utility shall be able to descend to arbitrary depths in a file hierarchy, and shall not fail
31718 due to path length limitations (unless an operand specified by the user exceeds system

31719 limitations).

31720 **OPTIONS**

31721 The *rm* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
31722 Utility Syntax Guidelines.

31723 The following options shall be supported:

31724 **–f** Do not prompt for confirmation. Do not write diagnostic messages or modify the
31725 exit status in the case of nonexistent operands. Any previous occurrences of the **–i**
31726 option shall be ignored.

31727 **–i** Prompt for confirmation as described previously. Any previous occurrences of the
31728 **–f** option shall be ignored.

31729 **–R** Remove file hierarchies. See the DESCRIPTION.

31730 **–r** Equivalent to **–R**.

31731 **OPERANDS**

31732 The following operand shall be supported:

31733 *file* A pathname of a directory entry to be removed.

31734 **STDIN**

31735 The standard input shall be used to read an input line in response to each prompt specified in |
31736 the STDOUT section. Otherwise, the standard input shall not be used. |

31737 **INPUT FILES**

31738 None.

31739 **ENVIRONMENT VARIABLES**

31740 The following environment variables shall affect the execution of *rm*:

31741 *LANG* Provide a default value for the internationalization variables that are unset or null.
31742 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
31743 Internationalization Variables for the precedence of internationalization variables
31744 used to determine the values of locale categories.)

31745 *LC_ALL* If set to a non-empty string value, override the values of all the other
31746 internationalization variables.

31747 *LC_COLLATE*
31748 Determine the locale for the behavior of ranges, equivalence classes, and multi-
31749 character collating elements used in the extended regular expression defined for
31750 the **yesexpr** locale keyword in the *LC_MESSAGES* category.

31751 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
31752 characters (for example, single-byte as opposed to multi-byte characters in
31753 arguments) and the behavior of character classes within regular expressions used
31754 in the extended regular expression defined for the **yesexpr** locale keyword in the
31755 *LC_MESSAGES* category.

31756 *LC_MESSAGES*
31757 Determine the locale for the processing of affirmative responses that should be
31758 used to affect the format and contents of diagnostic messages written to standard
31759 error.

31760 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

31761 **ASYNCHRONOUS EVENTS**
31762 Default.

31763 **STDOUT**
31764 Not used.

31765 **STDERR**
31766 Prompts shall be written to standard error under the conditions specified in the DESCRIPTION
31767 and OPTIONS sections. The prompts shall contain the *file* pathname, but their format is
31768 otherwise unspecified. The standard error also shall be used for diagnostic messages.

31769 **OUTPUT FILES**
31770 None.

31771 **EXTENDED DESCRIPTION**
31772 None.

31773 **EXIT STATUS**
31774 The following exit values shall be returned:

31775 0 All of the named directory entries for which *rm* performed actions equivalent to *rmdir*( ) or
31776 *unlink*( ) functions were removed.

31777 >0 An error occurred.

31778 **CONSEQUENCES OF ERRORS**
31779 Default.

31780 **APPLICATION USAGE**
31781 The *rm* utility is forbidden to remove the names dot and dot-dot in order to avoid the
31782 consequences of inadvertently doing something like:

31783 `rm –r .*`

31784 Some implementations do not permit the removal of the last link to an executable binary file that |
31785 is being executed; see the [EBUSY] error in the *unlink*( ) function defined in the System Interfaces
31786 volume of IEEE Std 1003.1-200x. Thus, the *rm* utility can fail to remove such files.

31787 The **–i** option causes *rm* to prompt and read the standard input even if the standard input is not
31788 a terminal, but in the absence of **–i** the mode prompting is not done when the standard input is
31789 not a terminal.

31790 **EXAMPLES**

31791 1. The following command:

31792 `rm a.out core`

31793 removes the directory entries: **a.out** and **core**.

31794 2. The following command:

31795 `rm –Rf junk`

31796 removes the directory **junk** and all its contents, without prompting.

31797 **RATIONALE**
31798 For absolute clarity, paragraphs (2b) and (3) in the DESCRIPTION of *rm* describing the behavior
31799 when prompting for confirmation, should be interpreted in the following manner:

```
31800 if ((NOT f_option) AND
31801     ((not_writable AND input_is_terminal) OR i_option))
```

The exact format of the interactive prompts is unspecified. Only the general nature of the contents of prompts are specified because implementations may desire more descriptive prompts than those used on historical implementations. Therefore, an application not using the **–f** option, or using the **–i** option, relies on the system to provide the most suitable dialog directly with the user, based on the behavior specified.

The **–r** option is historical practice on all known systems. The synonym **–R** option is provided for consistency with the other utilities in this volume of IEEE Std 1003.1-200x that provide options requesting recursive descent through the file hierarchy.

The behavior of the **–f** option in historical versions of *rm* is inconsistent. In general, along with ''forcing'' the unlink without prompting for permission, it always causes diagnostic messages to be suppressed and the exit status to be unmodified for nonexistent operands and files that cannot be unlinked. In some versions, however, the **–f** option suppresses usage messages and system errors as well. Suppressing such messages is not a service to either shell scripts or users.

It is less clear that error messages regarding files that cannot be unlinked (removed) should be suppressed. Although this is historical practice, this volume of IEEE Std 1003.1-200x does not permit the **–f** option to suppress such messages.

When given the **–r** and **–i** options, historical versions of *rm* prompt the user twice for each directory, once before removing its contents and once before actually attempting to delete the directory entry that names it. This allows the user to ''prune'' the file hierarchy walk. Historical versions of *rm* were inconsistent in that some did not do the former prompt for directories named on the command line and others had obscure prompting behavior when the **–i** option was specified and the permissions of the file did not permit writing. The POSIX Shell and Utilities *rm* differs little from historic practice, but does require that prompts be consistent. Historical versions of *rm* were also inconsistent in that prompts were done to both standard output and standard error. This volume of IEEE Std 1003.1-200x requires that prompts be done to standard error, for consistency with *cp* and *mv*, and to allow historical extensions to *rm* that provide an option to list deleted files on standard output.

The *rm* utility is required to descend to arbitrary depths so that any file hierarchy may be deleted. This means, for example, that the *rm* utility cannot run out of file descriptors during its descent (that is, if the number of file descriptors is limited, *rm* cannot be implemented in the historical fashion where one file descriptor is used per directory level). Also, *rm* is not permitted to fail because of path length restrictions, unless an operand specified by the user is longer than {PATH_MAX}.

The *rm* utility removes symbolic links themselves, not the files they refer to, as a consequence of the dependence on the *unlink*( ) functionality, per the DESCRIPTION. When removing hierarchies with **–r** or **–R**, the prohibition on following symbolic links has to be made explicit.

FUTURE DIRECTIONS

None.

SEE ALSO

rmdir, the System Interfaces volume of IEEE Std 1003.1-200x, *remove*( ), *unlink*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

FUTURE DIRECTIONS section added.

Issue 6

Text is added to clarify actions relating to symbolic links as specified in the IEEE P1003.2b draft standard.

NAME

rmdel — remove a delta from an SCCS file (**DEVELOPMENT**)

SYNOPSIS

XSI
```
rmdel –r SID file...
```

DESCRIPTION

The *rmdel* utility shall remove the delta specified by the SID from each named SCCS file. The delta to be removed shall be the most recent delta in its branch in the delta chain of each named SCCS file. In addition, the application shall ensure that the SID specified is not that of a version being edited for the purpose of making a delta; that is, if a *p-file* (see *get* (on page 2675)) exists for the named SCCS file, the SID specified shall not appear in any entry of the *p-file*.

Removal of a delta shall be restricted to:

1. The user who made the delta
2. The owner of the SCCS file
3. The owner of the directory containing the SCCS file

OPTIONS

The *rmdel* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following option shall be supported:

–r *SID* Specify the SCCS identification string (*SID*) of the delta to be deleted.

OPERANDS

The following operand shall be supported:

file A pathname of an existing SCCS file or a directory. If *file* is a directory, the *rmdel* utility shall behave as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with **s.**) and unreadable files shall be silently ignored.

If exactly one *file* operand appears, and it is '–', the standard input shall be read; each line of the standard input is taken to be the name of an SCCS file to be processed. Non-SCCS files and unreadable files shall be silently ignored.

STDIN

The standard input shall be a text file used only when the *file* operand is specified as '–'. Each line of the text file shall be interpreted as an SCCS pathname.

INPUT FILES

The SCCS files shall be files of unspecified format.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *rmdel*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
Not used.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
The SCCS files shall be files of unspecified format. During processing of a *file*, a temporary *x-file*, as described in *admin* (on page 2328), may be created and deleted; a locking *z-file*, as described in *get* (on page 2675), may be created and deleted.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
None.

EXAMPLES
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
delta, *get*, *prs*

CHANGE HISTORY
First released in Issue 2.

Issue 6
The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

The normative text is reworded to emphasize the term ‘‘shall’’ for implementation requirements.

NAME

rmdir — remove directories

SYNOPSIS

```
rmdir [–p] dir...
```

DESCRIPTION

The *rmdir* utility shall remove the directory entry specified by each *dir* operand, which, in order to succeed, the application shall ensure refers to an empty directory.

Directories shall be processed in the order specified. If a directory and a subdirectory of that directory are specified in a single invocation of the *rmdir* utility, the application shall specify the subdirectory before the parent directory so that the parent directory will be empty when the *rmdir* utility tries to remove it.

OPTIONS

The *rmdir* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following option shall be supported:

–p Remove all directories in a pathname. For each *dir* operand:

1. The directory entry it names shall be removed.
2. If the *dir* operand includes more than one pathname component, effects equivalent to the following command shall occur:

   ```
   rmdir –p $(dirname dir)
   ```

OPERANDS

The following operand shall be supported:

dir A pathname of an empty directory to be removed.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *rmdir*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

31973 **ASYNCHRONOUS EVENTS**
31974 Default.

31975 **STDOUT**
31976 Not used.

31977 **STDERR**
31978 The standard error shall be used only for diagnostic messages.

31979 **OUTPUT FILES**
31980 None.

31981 **EXTENDED DESCRIPTION**
31982 None.

31983 **EXIT STATUS**
31984 The following exit values shall be returned:

31985 0 Each directory entry specified by a *dir* operand was removed successfully.

31986 >0 An error occurred.

31987 **CONSEQUENCES OF ERRORS**
31988 Default.

31989 **APPLICATION USAGE**
31990 The definition of an empty directory is one that contains, at most, directory entries for dot and
31991 dot-dot.

31992 **EXAMPLES**
31993 If a directory **a** in the current directory is empty except it contains a directory **b** and **a/b** is empty
31994 except it contains a directory **c**:

31995
```
rmdir –p a/b/c
```

31996 removes all three directories.

31997 **RATIONALE**
31998 On historical System V systems, the **–p** option also caused a message to be written to the
31999 standard output. The message indicated whether the whole path was removed or whether part
32000 of the path remained for some reason. The STDERR section requires this diagnostic when the
32001 entire path specified by a *dir* operand is not removed, but does not allow the status message
32002 reporting success to be written as a diagnostic.

32003 The *rmdir* utility on System V also included an **–s** option that suppressed the informational
32004 message output by the **–p** option. This option has been omitted because the informational
32005 message is not specified by this volume of IEEE Std 1003.1-200x.

32006 **FUTURE DIRECTIONS**
32007 None.

32008 **SEE ALSO**
32009 *rm*, the System Interfaces volume of IEEE Std 1003.1-200x, *remove*( ), *rmdir*( ), *unlink*( )

32010 **CHANGE HISTORY**
32011 First released in Issue 2.

32012 **Issue 6**
32013 The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

NAME

sact — print current SCCS file-editing activity (**DEVELOPMENT**)

SYNOPSIS

XSI
```
sact file...
```

DESCRIPTION

The *sact* utility shall inform the user of any impending deltas to a named SCCS file by writing a list to standard output. This situation occurs when *get* **–e** has been executed previously without a subsequent execution of *delta*, *unget*, or *sccs* **unedit**.

OPTIONS

None.

OPERANDS

The following operand shall be supported:

file A pathname of an existing SCCS file or a directory. If *file* is a directory, the *sact* utility shall behave as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the pathname does not begin with **s.**) and unreadable files shall be silently ignored.

If exactly one *file* operand appears, and it is `'–'`, the standard input shall be read; each line of the standard input shall be taken to be the name of an SCCS file to be processed. Non-SCCS files and unreadable files shall be silently ignored.

STDIN

The standard input shall be a text file used only when the *file* operand is specified as `'–'`. Each line of the text file shall be interpreted as an SCCS pathname.

INPUT FILES

Any SCCS files interrogated are files of an unspecified format.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *sact*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

32056 **STDOUT**

32057 The output for each named file shall consist of a line in the following format:

32058 `"%s␣%s␣%s␣%s␣%s\n", <SID>, <new SID>, <login>, <date>, <time>`

32059 <*SID*> Specifies the SID of a delta that currently exists in the SCCS file to which changes
32060 are made to make the new delta.

32061 <*new SID*> Specifies the SID for the new delta to be created.

32062 <*login*> Contains the login name of the user who makes the delta (that is, who executed a
32063 *get* for editing).

32064 <*date*> Contains the date that *get* **–e** was executed, in the format used by the *prs* **:D:** data
32065 keyword.

32066 <*time*> Contains the time that *get* **–e** was executed, in the format used by the *prs* **:T:** data
32067 keyword.

32068 If there is more than one named file or if a directory or standard input is named, each pathname
32069 shall be written before each of the preceding lines:

32070 `"\n%s:\n", <pathname>`

32071 **STDERR**

32072 The standard error shall be used only for optional informative messages concerning SCCS files
32073 with no impending deltas, and for diagnostic messages.

32074 **OUTPUT FILES**

32075 None.

32076 **EXTENDED DESCRIPTION**

32077 None.

32078 **EXIT STATUS**

32079 The following exit values shall be returned:

32080 0 Successful completion.

32081 >0 An error occurred.

32082 **CONSEQUENCES OF ERRORS**

32083 Default.

32084 **APPLICATION USAGE**

32085 None.

32086 **EXAMPLES**

32087 None.

32088 **RATIONALE**

32089 None.

32090 **FUTURE DIRECTIONS**

32091 None.

32092 **SEE ALSO**

32093 *delta*, *get*, *unget*

CHANGE HISTORY

First released in Issue 2.

Issue 6

The normative text is reworded to emphasize the term ‘‘shall’’ for implementation requirements.

NAME

sccs — front end for the SCCS subsystem (**DEVELOPMENT**)

SYNOPSIS

XSI
```
sccs [–r][–d path][–p path] command [options...][operands...]
```

DESCRIPTION

The *sccs* utility is a front end to the SCCS programs. It also includes the capability to run set-user-id to another user to provide additional protection.

The *sccs* utility shall invoke the specified *command* with the specified *options* and *operands*. By default, each of the *operands* shall be modified by prefixing it with the string "SCCS/s.".

The *command* can be the name of one of the SCCS utilities in this volume of IEEE Std 1003.1-200x (*admin*, *delta*, *get*, *prs*, *rmdel*, *sact*, *unget*, *val*, or *what*) or one of the pseudo-utilities listed in the EXTENDED DESCRIPTION section.

OPTIONS

The *sccs* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines, except that *options* operands are actually options to be passed to the utility named by *command*. When the portion of the command:

command **[***options ...* **] [***operands ...* **]**

is considered, all of the pseudo-utilities used as *command* shall support the Utility Syntax Guidelines. Any of the other SCCS utilities that can be invoked in this manner support the Guidelines to the extent indicated by their individual OPTIONS sections.

The following options shall be supported preceding the *command* operand:

–d *path* A pathname of a directory to be used as a root directory for the SCCS files. The default shall be the current directory. The **–d** option shall take precedence over the *PROJECTDIR* variable. See **–p**.

–p *path* A pathname of a directory in which the SCCS files are located. The default shall be the **SCCS** directory.

The **–p** option differs from the **–d** option in that the **–d** option-argument shall be prefixed to the entire pathname and the **–p** option-argument shall be inserted before the final component of the pathname. For example:

```
sccs –d /x –p y get a/b
```

converts to:

```
get /x/a/y/s.b
```

This allows the creation of aliases such as:

```
alias syssccs="sccs –d /usr/src"
```

which is used as:

```
syssccs get cmd/who.c
```

–r Invoke *command* with the real user ID of the process, not any effective user ID that the *sccs* utility is set to. Certain commands (*admin*, **check**, **clean**, **diffs**, **info**, *rmdel*, and **tell**) cannot be run set-user-ID by all users, since this would allow anyone to change the authorizations. These commands are always run as the real user.

32139 **OPERANDS**

32140 The following operands shall be supported:

32141 *command* An SCCS utility name or the name of one of the pseudo-utilities listed in the
32142 EXTENDED DESCRIPTION section.

32143 *options* An option or option-argument to be passed to *command*.

32144 *operands* An operand to be passed to *command*.

32145 **STDIN**

32146 See the utility description for the specified *command*.

32147 **INPUT FILES**

32148 See the utility description for the specified *command*.

32149 **ENVIRONMENT VARIABLES**

32150 The following environment variables shall affect the execution of *sccs*:

32151 *LANG* Provide a default value for the internationalization variables that are unset or null.
32152 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
32153 Internationalization Variables for the precedence of internationalization variables
32154 used to determine the values of locale categories.)

32155 *LC_ALL* If set to a non-empty string value, override the values of all the other
32156 internationalization variables.

32157 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
32158 characters (for example, single-byte as opposed to multi-byte characters in
32159 arguments and input files).

32160 *LC_MESSAGES*
32161 Determine the locale that should be used to affect the format and contents of
32162 diagnostic messages written to standard error.

32163 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

32164 *PROJECTDIR*
32165 Provide a default value for the **−d** *path* option. If the value of *PROJECTDIR* begins
32166 with a slash, it shall be considered an absolute pathname; otherwise, the value of
32167 *PROJECTDIR* is treated as a user name and that user's initial working directory
32168 shall be examined for a subdirectory **src** or **source**. If such a directory is found, it
32169 shall be used. Otherwise, the value shall be used as a relative pathname.

32170 Additional environment variable effects may be found in the utility description for the specified
32171 *command*.

32172 **ASYNCHRONOUS EVENTS**

32173 Default.

32174 **STDOUT**

32175 See the utility description for the specified *command*.

32176 **STDERR**

32177 See the utility description for the specified *command*.

32178 **OUTPUT FILES**

32179 See the utility description for the specified *command*.

EXTENDED DESCRIPTION

The following pseudo-utilities shall be supported as *command* operands. All options referred to in the following list are values given in the *options* operands following *command*.

check Equivalent to **info**, except that nothing shall be printed if nothing is being edited, and a non-zero exit status shall be returned if anything is being edited. The intent is to have this included in an ''install'' entry in a makefile to ensure that everything is included into the SCCS file before a version is installed.

clean Remove everything from the current directory that can be recreated from SCCS files, but do not remove any files being edited. If the **–b** option is given, branches shall be ignored in the determination of whether they are being edited; this is dangerous if branches are kept in the same directory.

create Create an SCCS file, taking the initial contents from the file of the same name. Any options to *admin* are accepted. If the creation is successful, the original files shall be renamed by prefixing the basenames with a comma. These renamed files should be removed after it has been verified that the SCCS files have been created successfully.

delget Perform a *delta* on the named files and then *get* new versions. The new versions shall have ID keywords expanded and shall not be editable. Any **–m**, **–p**, **–r**, **–s**, and **–y** options shall be passed to *delta*, and any **–b**, **–c**, **–e**, **–i**, **–k**, **–l**, **–s**, and **–x** options shall be passed to *get*.

deledit Equivalent to **delget**, except that the *get* phase shall include the **–e** option. This option is useful for making a checkpoint of the current editing phase. The same options shall be passed to *delta* as described above, and all the options listed for *get* above except **–e** shall be passed to **edit**.

diffs Write a difference listing between the current version of the files checked out for editing and the versions in SCCS format. Any **–r**, **–c**, **–i**, **–x**, and **–t** options shall be passed to *get*; any **–l**, **–s**, **–e**, **–f**, **–h**, and **–b** options shall be passed to *diff*. A **–C** option shall be passed to *diff* as **–c**.

edit Equivalent to *get* **–e**.

fix Remove the named delta, but leave a copy of the delta with the changes that were in it. It is useful for fixing small compiler bugs, and so on. The application shall ensure that it is followed by a **–r** *SID* option. Since **fix** does not leave audit trails, it should be used carefully.

info Write a listing of all files being edited. If the **–b** option is given, branches (that is, SIDs with two or fewer components) shall be ignored. If a **–u** *user* option is given, then only files being edited by the named user shall be listed. A **–U** option shall be equivalent to **–u**<*current user*>.

print Write out verbose information about the named files, equivalent to *sccs prs*.

tell Write a <newline>-separated list of the files being edited to standard output. Takes the **–b**, **–u**, and **–U** options like **info** and **check**.

unedit This is the opposite of an **edit** or a *get* **–e**. It should be used with caution, since any changes made since the *get* are lost.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Many of the SCCS utilities take directory names as operands as well as specific filenames. The pseudo-utilities supported by *sccs* are not described as having this capability, but are not prohibited from doing so.

EXAMPLES

1. To get a file for editing, edit it and produce a new delta:

```
sccs get –e file.c
ex file.c
sccs delta file.c
```

2. To get a file from another directory:

```
sccs –p /usr/src/sccs/s. get cc.c
```

or:

```
sccs get /usr/src/sccs/s.cc.c
```

3. To make a delta of a large number of files in the current directory:

```
sccs delta *.c
```

4. To get a list of files being edited that are not on branches:

```
sccs info –b
```

5. To delta everything being edited by the current user:

```
sccs delta $(sccs tell –U)
```

6. In a makefile, to get source files from an SCCS file if it does not already exist:

```
SRCS = <list of source files>
$(SRCS):
    sccs get $(REL) $@
```

RATIONALE

SCCS and its associated utilities are part of the XSI Development Utilities option within the XSI extension.

SCCS is an abbreviation for Source Code Control System. It is a maintenance and enhancement tracking tool. When a file is put under SCCS, the source code control system maintains the file and, when changes are made, identifies and stores them in the file with the original source code and/or documentation. As other changes are made, they too are identified and retained in the file.

Retrieval of the original and any set of changes is possible. Any version of the file as it develops can be reconstructed for inspection or additional modification. History data can be stored with each version, documenting why the changes were made, who made them, and when they were made.

FUTURE DIRECTIONS

None.

SEE ALSO

admin, *delta*, *get*, *make*, *prs*, *rmdel*, *sact*, *unget*, *val*, *what*

CHANGE HISTORY

First released in Issue 4.

Issue 6

In the ENVIRONMENT VARIABLES section, the *PROJECTDIR* description is updated from ‘‘otherwise, the home directory of a user of that name is examined’’ to ‘‘otherwise, the value of *PROJECTDIR* is treated as a user name and that user’s initial working directory is examined’’.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

The normative text is reworded to emphasize the term ‘‘shall’’ for implementation requirements.

NAME

sed — stream editor

SYNOPSIS

```
sed [–n] script[file...]
sed [–n][–e script]...[–f script_file]...[file...]
```

DESCRIPTION

The *sed* utility is a stream editor that shall read one or more text files, make editing changes according to a script of editing commands, and write the results to standard output. The script shall be obtained from either the *script* operand string or a combination of the option-arguments from the **–e** *script* and **–f** *script_file* options.

OPTIONS

The *sed* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines, except that the order of presentation of the **–e** and **–f** options is significant.

The following options shall be supported:

–e *script* Add the editing commands specified by the *script* option-argument to the end of the script of editing commands. The *script* option-argument shall have the same properties as the *script* operand, described in the OPERANDS section.

–f *script_file* Add the editing commands in the file *script_file* to the end of the script.

–n Suppress the default output (in which each line, after it is examined for editing, is written to standard output). Only lines explicitly selected for output are written.

Multiple **–e** and **–f** options may be specified. All commands shall be added to the script in the order specified, regardless of their origin.

OPERANDS

The following operands shall be supported:

file A pathname of a file whose contents are read and edited. If multiple *file* operands are specified, the named files shall be read in the order specified and the concatenation shall be edited. If no *file* operands are specified, the standard input shall be used.

script A string to be used as the script of editing commands. The application shall not present a *script* that violates the restrictions of a text file except that the final character need not be a <newline>.

STDIN

The standard input shall be used only if no *file* operands are specified. See the INPUT FILES section.

INPUT FILES

The input files shall be text files. The *script_file*s named by the **–f** option shall consist of editing commands.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *sed*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes, and multi-character collating elements within regular expressions.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), and the behavior of character classes within regular expressions.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The input files shall be written to standard output, with the editing commands specified in the script applied. If the **–n** option is specified, only those input lines selected by the script shall be written to standard output.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

The output files shall be text files whose formats are dependent on the editing commands given.

EXTENDED DESCRIPTION

The *script* shall consist of editing commands of the following form:

[*address***[,***address***]]***function*

where *function* represents a single-character command verb from the list in **Editing Commands in sed** (on page 3041), followed by any applicable arguments.

The command can be preceded by <blank>s and/or semicolons. The function can be preceded by <blank>s. These optional characters shall have no effect.

In default operation, *sed* cyclically shall append a line of input, less its terminating <newline>, into the pattern space. Normally the pattern space will be empty, unless a **D** command terminated the last cycle. The *sed* utility shall then apply in sequence all commands whose addresses select that pattern space, and at the end of the script copy the pattern space to standard output (except when **–n** is specified) and delete the pattern space. Whenever the pattern space is written to standard output or a named file, *sed* shall immediately follow it with a <newline>.

Some of the editing commands use a hold space to save all or part of the pattern space for subsequent retrieval. The pattern and hold spaces shall each be able to hold at least 8 192 bytes.

32357 **Addresses in sed**

32358 An address is either a decimal number that counts input lines cumulatively across files, a `'$'`
32359 character that addresses the last line of input, or a context address (which consists of a BRE, as
32360 described in **Regular Expressions in sed**, preceded and followed by a delimiter, usually a slash).

32361 An editing command with no addresses shall select every pattern space.

32362 An editing command with one address shall select each pattern space that matches the address.

32363 An editing command with two addresses shall select the inclusive range from the first pattern
32364 space that matches the first address through the next pattern space that matches the second. (If
32365 the second address is a number less than or equal to the line number first selected, only one line
32366 shall be selected.) Starting at the first line following the selected range, *sed* shall look again for
32367 the first address. Thereafter, the process shall be repeated. Omitting either or both of the address
32368 components in the following form produces undefined results:

32369 **[***address***[,***address***]]**

32370 **Regular Expressions in sed**

32371 The *sed* utility shall support the BREs described in the Base Definitions volume of
32372 IEEE Std 1003.1-200x, Section 9.3, Basic Regular Expressions, with the following additions:

32373 • In a context address, the construction `"\cBREc"`, where *c* is any character other than
32374 backslash or <newline>, shall be identical to `"/BRE/"`. If the character designated by *c*
32375 appears following a backslash, then it shall be considered to be that literal character, which
32376 shall not terminate the BRE. For example, in the context address `"\xabc\xdefx"`, the
32377 second *x* stands for itself, so that the BRE is `"abcxdef"`.

32378 • The escape sequence `'\n'` shall match a <newline> embedded in the pattern space. A literal
32379 <newline> shall not be used in the BRE of a context address or in the substitute function.

32380 • If an RE is empty (that is, no pattern is specified) *sed* shall behave as if the last RE used in the
32381 last command applied (either as an address or as part of a substitute command) was
32382 specified.

32383 **Editing Commands in sed**

32384 In the following list of editing commands, the maximum number of permissible addresses for
32385 each function is indicated by [*0addr*], [*1addr*], or [*2addr*], representing zero, one, or two
32386 addresses.

32387 The argument *text* shall consist of one or more lines. Each embedded <newline> in the text shall
32388 be preceded by a backslash. Other backslashes in text shall be removed, and the following
32389 character shall be treated literally.

32390 The **r** and **w** command verbs, and the *w* flag to the **s** command, take an optional *rfile* (or *wfile*)
32391 parameter, separated from the command verb letter or flag by one or more <blank>s;
32392 implementations may allow zero separation as an extension.

32393 The argument *rfile* or the argument *wfile* shall terminate the editing command. Each *wfile* shall be
32394 created before processing begins. Implementations shall support at least ten *wfile* arguments in
32395 the script; the actual number (greater than or equal to 10) that is supported by the |
32396 implementation is unspecified. The use of the *wfile* parameter shall cause that file to be initially |
32397 created, if it does not exist, or shall replace the contents of an existing file.

32398 The **b**, **r**, **s**, **t**, **w**, **y**, and **:** command verbs shall accept additional arguments. The following
32399 synopses indicate which arguments shall be separated from the command verbs by a single

<space>.

The **a** and **r** commands schedule text for later output. The text specified for the **a** command, and the contents of the file specified for the **r** command, shall be written to standard output just before the next attempt to fetch a line of input when executing the **N** or **n** commands, or when reaching the end of the script. If written when reaching the end of the script, and the **–n** option was not specified, the text shall be written after copying the pattern space to standard output. The contents of the file specified for the **r** command shall be as of the time the output is written, not the time the **r** command is applied. The text shall be output in the order in which the **a** and **r** commands were applied to the input.

Command verbs other than **{**, **a**, **b**, **c**, **i**, **r**, **t**, **w**, **:**, and **#** can be followed by a semicolon, optional <blank>s, and another command verb. However, when the **s** command verb is used with the *w* flag, following it with another command in this manner produces undefined results.

A function can be preceded by one or more '**!**' characters, in which case the function shall be applied if the addresses do not select the pattern space. Zero or more <blank>s shall be accepted before the first '**!**' character. It is unspecified whether <blank>s can follow a '**!**' character, and conforming applications shall not follow a '**!**' character with <blank>s.

[*2addr***] {***function*
function
...
} Execute a list of *sed* functions only when the pattern space is selected. The list of *sed* functions shall be surrounded by braces and separated by <newline>s, and conform to the following rules. The braces can be preceded or followed by <blank>s. The functions can be preceded by <blank>s, but shall not be followed by <blank>s. The <right-brace> shall be preceded by a <newline> and can be preceded or followed by <blank>s.

[*1addr***]a**
text Write text to standard output as described previously.

[*2addr***]b [***label***]**
Branch to the **:** function bearing the *label*. If *label* is not specified, branch to the end of the script. The implementation shall support *label*s recognized as unique up to at least 8 characters; the actual length (greater than or equal to 8) that shall be supported by the implementation is unspecified. It is unspecified whether exceeding a label length causes an error or a silent truncation.

[*2addr***]c**
text Delete the pattern space. With a 0 or 1 address or at the end of a 2-address range, place *text* on the output and start the next cycle.

[*2addr***]d** Delete the pattern space and start the next cycle.

[*2addr***]D** Delete the initial segment of the pattern space through the first <newline> and start the next cycle.

[*2addr***]g** Replace the contents of the pattern space by the contents of the hold space.

[*2addr***]G** Append to the pattern space a <newline> followed by the contents of the hold space.

[*2addr***]h** Replace the contents of the hold space with the contents of the pattern space.

[*2addr***]H** Append to the hold space a <newline> followed by the contents of the pattern space.

32445 **[***1addr***]i**
32446 *text* Write *text* to standard output.

32447 **[***2addr***]l** (The letter ell.) Write the pattern space to standard output in a visually
32448 unambiguous form. The characters listed in the Base Definitions volume of
32449 IEEE Std 1003.1-200x, Table 5-1, Escape Sequences and Associated Actions (`'\\'`,
32450 `'\a'`, `'\b'`, `'\f'`, `'\r'`, `'\t'`, `'\v'`) shall be written as the corresponding
32451 escape sequence; the `'\n'` in that table is not applicable. Non-printable characters
32452 not in that table shall be written as one three-digit octal number (with a preceding
32453 backslash) for each byte in the character (most significant byte first). If the size of a
32454 byte on the system is greater than 9 bits, the format used for non-printable
32455 characters is implementation-defined.

32456 Long lines shall be folded, with the point of folding indicated by writing a
32457 backslash followed by a <newline>; the length at which folding occurs is
32458 unspecified, but should be appropriate for the output device. The end of each line
32459 shall be marked with a `'$'`.

32460 **[***2addr***]n** Write the pattern space to standard output if the default output has not been
32461 suppressed, and replace the pattern space with the next line of input, less its
32462 terminating <newline>.

32463 If no next line of input is available, the **n** command verb shall branch to the end of
32464 the script and quit without starting a new cycle.

32465 **[***2addr***]N** Append the next line of input, less its terminating <newline>, to the pattern space, |
32466 using an embedded <newline> to separate the appended material from the original |
32467 material. Note that the current line number changes.

32468 If no next line of input is available, the **N** command verb shall branch to the end of |
32469 the script and quit without starting a new cycle or copying the pattern space to
32470 standard output.

32471 **[***2addr***]p** Write the pattern space to standard output.

32472 **[***2addr***]P** Write the pattern space, up to the first <newline>, to standard output.

32473 **[***1addr***]q** Branch to the end of the script and quit without starting a new cycle.

32474 **[***1addr***]r** *rfile* Copy the contents of *rfile* to standard output as described previously. If *rfile* does
32475 not exist or cannot be read, it shall be treated as if it were an empty file, causing no
32476 error condition.

32477 **[***2addr***]s/***BRE/replacement/flags*
32478 Substitute the replacement string for instances of the BRE in the pattern space. Any
32479 character other than backslash or <newline> can be used instead of a slash to
32480 delimit the BRE and the replacement. Within the BRE and the replacement, the
32481 BRE delimiter itself can be used as a literal character if it is preceded by a
32482 backslash.

32483 The replacement string shall be scanned from beginning to end. An ampersand
32484 (`'&'`) appearing in the replacement shall be replaced by the string matching the
32485 BRE. The special meaning of `'&'` in this context can be suppressed by preceding it
32486 by a backslash. The characters "`\`*n*", where *n* is a digit, shall be replaced by the
32487 text matched by the corresponding backreference expression. The special meaning
32488 of "`\`*n*" where *n* is a digit in this context, can be suppressed by preceding it by a
32489 backslash. For each other backslash (`'\'`) encountered, the following character |
32490 shall lose its special meaning (if any). The meaning of a `'\'` immediately followed |

by any character other than '&', '\', a digit, or the delimiter character used for this command, is unspecified.

A line can be split by substituting a <newline> into it. The application shall escape the <newline> in the replacement by preceding it by a backslash. A substitution shall be considered to have been performed even if the replacement string is identical to the string that it replaces. Any backslash used to alter the default meaning of a subsequent character shall be discarded from the BRE or the replacement before evaluating the BRE or using the replacement.

The value of *flags* shall be zero or more of:

n Substitute for the *n*th occurrence only of the BRE found within the pattern space.

g Globally substitute for all non-overlapping instances of the BRE rather than just the first one. If both **g** and **n** are specified, the results are unspecified.

p Write the pattern space to standard output if a replacement was made.

w *wfile* Write. Append the pattern space to *wfile* if a replacement was made. A conforming application shall precede the *wfile* argument with one or more <blank>s. If the *w* flag is not the last flag value given in a concatenation of multiple flag values, the results are undefined.

[*2addr***]t [***label***]**

Test. Branch to the **:** command verb bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t**. If *label* is not specified, branch to the end of the script.

[*2addr***]w** *wfile*

Append (write) the pattern space to *wfile*.

[*2addr***]x** Exchange the contents of the pattern and hold spaces.

[*2addr***]y/***string1***/***string2***/**

Replace all occurrences of characters in *string1* with the corresponding characters in *string2*. If a backslash followed by an 'n' appear in *string1* or *string2*, the two characters shall be handled as a single <newline>. If the number of characters in *string1* and *string2* are not equal, or if any of the characters in *string1* appear more than once, the results are undefined. Any character other than backslash or <newline> can be used instead of slash to delimit the strings. If the delimiter is not *n*, within *string1* and *string2*, the delimiter itself can be used as a literal character if it is preceded by a backslash. If a backslash character is immediately followed by a backslash character in *string1* or *string2*, the two backslash characters shall be counted as a single literal backslash character. The meaning of a backslash followed by any character that is not 'n', a backslash, or the delimiter character is undefined.

[*0addr***]:***label* Do nothing. This command bears a *label* to which the **b** and **t** commands branch.

[*1addr***]=** Write the following to standard output:

"%d\n", *<current line number>*

[*0addr***]** Ignore this empty command.

[*0addr***]#** Ignore the '#' and the remainder of the line (treat them as a comment), with the single exception that if the first two characters in the script are "#n", the default output shall be suppressed; this shall be the equivalent of specifying **–n** on the command line.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Regular expressions match entire strings, not just individual lines, but a <newline> is matched by '\n' in a *sed* RE; a <newline> is not allowed by the general definition of regular expression in IEEE Std 1003.1-200x. Also note that '\n' cannot be used to match a <newline> at the end of an arbitrary input line; <newline>s appear in the pattern space as a result of the **N** editing command.

EXAMPLES

This *sed* script simulates the BSD *cat* **–s** command, squeezing excess blank lines from standard input.

```
sed –n '
# Write non-empty lines.
/./ {
    p
    d
    }
# Write a single empty line, then look for more empty lines.
/^$/    p
# Get next line, discard the held <newline> (empty line),
# and look for more empty lines.
:Empty
/^$/    {
    N
    s/.//
    b Empty
    }
# Write the non-empty line before going back to search
# for the first in a set of empty lines.
    p
'
```

RATIONALE

This volume of IEEE Std 1003.1-200x requires implementations to support at least ten distinct *wfile*s, matching historical practice on many implementations. Implementations are encouraged to support more, but conforming applications should not exceed this limit.

The exit status codes specified here are different from those in System V. System V returns 2 for garbled *sed* commands, but returns zero with its usage message or if the input file could not be opened. The standard developers considered this to be a bug.

The manner in which the **l** command writes non-printable characters was changed to avoid the historical backspace-overstrike method, and other requirements to achieve unambiguous output were added. See the RATIONALE for *ed* (on page 2537) for details of the format chosen, which is the same as that chosen for *sed*.

This volume of IEEE Std 1003.1-200x requires implementations to provide pattern and hold spaces of at least 8192 bytes, larger than the 4000 bytes spaces used by some historical implementations, but less than the 20480 bytes limit used in an early proposal. Implementations are encouraged to allocate dynamically larger pattern and hold spaces as needed.

The requirements for acceptance of <blank>s and <space>s in command lines has been made more explicit than in early proposals to describe clearly the historical practice and to remove confusion about the phrase ''protect initial blanks [*sic*] and tabs from the stripping that is done on every script line'' that appears in much of the historical documentation of the *sed* utility description of text. (Not all implementations are known to have stripped <blank>s from text lines, although they all have allowed leading <blank>s preceding the address on a command line.)

The treatment of `'#'` comments differs from the SVID which only allows a comment as the first line of the script, but matches BSD-derived implementations. The comment character is treated as a command, and it has the same properties in terms of being accepted with leading <blank>s; the BSD implementation has historically supported this.

Early proposals required that a *script_file* have at least one non-comment line. Some historical implementations have behaved in unexpected ways if this were not the case. The standard developers considered that this was incorrect behavior and that application developers should not have to avoid this feature. A correct implementation of this volume of IEEE Std 1003.1-200x shall permit *script_file*s that consist only of comment lines.

Early proposals indicated that if **–e** and **–f** options were intermixed, all **–e** options were processed before any **–f** options. This has been changed to process them in the order presented because it matches historical practice and is more intuitive.

The treatment of the **p** flag to the **s** command differs between System V and BSD-based systems when the default output is suppressed. In the two examples:

```
echo a | sed    's/a/A/p'
echo a | sed –n 's/a/A/p'
```

This volume of IEEE Std 1003.1-200x, BSD, System V documentation, and the SVID indicate that the first example should write two lines with **A**, whereas the second should write one. Some System V systems write the **A** only once in both examples because the **p** flag is ignored if the **–n** option is not specified.

This is a case of a diametrical difference between systems that could not be reconciled through the compromise of declaring the behavior to be unspecified. The SVID/BSD/System V documentation behavior was adopted for this volume of IEEE Std 1003.1-200x because:

- No known documentation for any historic system describes the interaction between the **p** flag and the **–n** option.
- The selected behavior is more correct as there is no technical justification for any interaction between the **p** flag and the **–n** option. A relationship between **–n** and the **p** flag might imply that they are only used together, but this ignores valid scripts that interrupt the cyclical nature of the processing through the use of the **D**, **d**, **q**, or branching commands. Such scripts rely on the **p** suffix to write the pattern space because they do not make use of the default output at the ''bottom'' of the script.

- Because the –**n** option makes the **p** flag unnecessary, any interaction would only be useful if *sed* scripts were written to run both with and without the –**n** option. This is believed to be unlikely. It is even more unlikely that programmers have coded the **p** flag expecting it to be unnecessary. Because the interaction was not documented, the likelihood of a programmer discovering the interaction and depending on it is further decreased.
- Finally, scripts that break under the specified behavior produce too much output instead of too little, which is easier to diagnose and correct.

The form of the substitute command that uses the **n** suffix was limited to the first 512 matches in an early proposal. This limit has been removed because there is no reason an editor processing lines of {LINE_MAX} length should have this restriction. The command **s/a/A/2047** should be able to substitute the 2 047th occurrence of **a** on a line.

The **b**, **t**, and **:** commands are documented to ignore leading white space, but no mention is made of trailing white space. Historical implementations of *sed* assigned different locations to the labels `'x'` and `"x "`. This is not useful, and leads to subtle programming errors, but it is historical practice, and changing it could theoretically break working scripts. Implementors are encouraged to provide warning messages about labels that are never used or jumps to labels that do not exist.

Historically, the *sed* **!** and **}** editing commands did not permit multiple commands on a single line using a semicolon as a command delimiter. Implementations are permitted, but not required, to support this extension.

FUTURE DIRECTIONS

None.

SEE ALSO

awk, *ed*, *grep*

CHANGE HISTORY

First released in Issue 2.

Issue 5

FUTURE DIRECTIONS section added.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- Implementations are required to support at least ten *wfile* arguments in an editing command.

The EXTENDED DESCRIPTION is changed to align with the IEEE P1003.2b draft standard.

IEEE PASC Interpretation 1003.2 #190 is applied. |

IEEE PASC Interpretation 1003.2 #203 is applied, clarifying the meaning of the backslash escape sequences in a replacement string for a BRE. | |

32663 **NAME**

32664 sh — shell, the standard command language interpreter

32665 **SYNOPSIS**

32666 `sh [`–abCefhimnuvx`][`–o *option*`][`+abCefhmnuvx`][`+o *option*`]`
32667 `[`*command_file* `[`*argument*`...]]`

32668 `sh –c[`–abCefhimnuvx`][`–o *option*`][`+abCefhimnuvx`][`+o *option*`]`*command_string*
32669 `[`*command_name* `[`*argument*`...]]`

32670 `sh –s[`–abCefhimnuvx`][`–o *option*`][`+abCefhimnuvx`][`+o *option*`][`*argument*`]`

32671 **DESCRIPTION**

32672 The *sh* utility is a command language interpreter that shall execute commands read from a
32673 command line string, the standard input, or a specified file. The application shall ensure that the
32674 commands to be executed are expressed in the language described in Chapter 2 (on page 2231).

32675 Pathname expansion shall not fail due to the size of a file.

32676 Shell input and output redirections have an implementation-defined offset maximum that is
32677 established in the open file description.

32678 **OPTIONS**

32679 The *sh* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
32680 Utility Syntax Guidelines, with an extension for support of a leading plus sign (`'+'`) as noted
32681 below.

32682 The **–a**, **–b**, **–C**, **–e**, **–f**, **–m**, **–n**, **–o** *option*, **–u**, **–v**, and **–x** options are described as part of the *set*
32683 utility in Section 2.14 (on page 2266). The option letters derived from the *set* special built-in shall
32684 also be accepted with a leading plus sign (`'+'`) instead of a leading hyphen (meaning the reverse
32685 case of the option as described in this volume of IEEE Std 1003.1-200x).

32686 The following additional options shall be supported:

32687 **–c** Read commands from the *command_string* operand. Set the value of special
32688 parameter 0 (see Section 2.5.2 (on page 2235)) from the value of the *command_name*
32689 operand and the positional parameters ($1, $2, and so on) in sequence from the
32690 remaining *argument* operands. No commands shall be read from the standard
32691 input.

32692 **–i** Specify that the shell is *interactive*; see below. An implementation may treat
32693 specifying the **–i** option as an error if the real user ID of the calling process does
32694 not equal the effective user ID or if the real group ID does not equal the effective
32695 group ID.

32696 **–s** Read commands from the standard input.

32697 If there are no operands and the **–c** option is not specified, the **–s** option shall be assumed.

32698 If the **–i** option is present, or if there are no operands and the shell's standard input and standard
32699 error are attached to a terminal, the shell is considered to be *interactive*.

32700 **OPERANDS**

32701 The following operands shall be supported:

32702 – A single hyphen shall be treated as the first operand and then ignored. If both `'–'` |
32703 and `"––"` are given as arguments, or if other operands precede the single hyphen,
32704 the results are undefined.

32705 *argument* The positional parameters ($1, $2, and so on) shall be set to *arguments*, if any.

command_file The pathname of a file containing commands. If the pathname contains one or more slash characters, the implementation attempts to read that file; the file need not be executable. If the pathname does not contain a slash character:

- The implementation shall attempt to read that file from the current working directory; the file need not be executable.
- If the file is not in the current working directory, the implementation may perform a search for an executable file using the value of *PATH*, as described in Section 2.9.1.1 (on page 2249).

Special parameter 0 (see Section 2.5.2 (on page 2235)) shall be set to the value of *command_file*. If *sh* is called using a synopsis form that omits *command_file*, special parameter 0 shall be set to the value of the first argument passed to *sh* from its parent (for example, *argv*[0] for a C program), which is normally a pathname used to execute the *sh* utility.

command_name
A string assigned to special parameter 0 when executing the commands in *command_string*. If *command_name* is not specified, special parameter 0 shall be set to the value of the first argument passed to *sh* from its parent (for example, *argv*[0] for a C program), which is normally a pathname used to execute the *sh* utility.

command_string
A string that shall be interpreted by the shell as one or more commands, as if the string were the argument to the *system*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x. If the *command_string* operand is an empty string, *sh* shall exit with a zero exit status.

STDIN

The standard input shall be used only if one of the following is true:

- The **–s** option is specified.
- The **–c** option is not specified and no operands are specified.
- The script executes one or more commands that require input from standard input (such as a *read* command that does not redirect its input).

See the INPUT FILES section.

When the shell is using standard input and it invokes a command that also uses standard input, the shell shall ensure that the standard input file pointer points directly after the command it has read when the command begins execution. It shall not read ahead in such a manner that any characters intended to be read by the invoked command are consumed by the shell (whether interpreted by the shell or not) or that characters that are not read by the invoked command are not seen by the shell. When the command expecting to read standard input is started asynchronously by an interactive shell, it is unspecified whether characters are read by the command or interpreted by the shell.

If the standard input to *sh* is a FIFO or terminal device and is set to non-blocking reads, then *sh* shall enable blocking reads on standard input. This shall remain in effect when the command completes.

INPUT FILES

The input file shall be a text file, except that line lengths shall be unlimited. If the input file is empty or consists solely of blank lines or comments, or both, *sh* shall exit with a zero exit status.

32750 **ENVIRONMENT VARIABLES**

32751 The following environment variables shall affect the execution of *sh*:

32752 *ENV* This variable, when and only when an interactive shell is invoked, shall be
32753 subjected to parameter expansion (see Section 2.6.2 (on page 2239)) by the shell,
32754 and the resulting value shall be used as a pathname of a file containing shell
32755 commands to execute in the current environment. The file need not be executable.
32756 If the expanded value of *ENV* is not an absolute pathname, the results are
32757 unspecified. *ENV* shall be ignored if the real and effective user IDs or real and
32758 effective group IDs of the process are different.

32759 *FCEDIT* This variable, when expanded by the shell, shall determine the default value for |
32760 the **–e** *editor* option's *editor* option-argument. If *FCEDIT* is null or unset, *ed* shall be |
32761 used as the editor. This volume of IEEE Std 1003.1-200x specifies the effects of this
32762 variable only for systems supporting the User Portability Utilities option.

32763 *HISTFILE* Determine a pathname naming a command history file. If the *HISTFILE* variable is
32764 not set, the shell may attempt to access or create a file **.sh_history** in the directory
32765 referred to by the *HOME* environment variable. If the shell cannot obtain both read
32766 and write access to, or create, the history file, it shall use an unspecified
32767 mechanism that allows the history to operate properly. (References to history
32768 ''file'' in this section shall be understood to mean this unspecified mechanism in
32769 such cases.) An implementation may choose to access this variable only when
32770 initializing the history file; this initialization shall occur when *fc* or *sh* first attempt
32771 to retrieve entries from, or add entries to, the file, as the result of commands issued
32772 by the user, the file named by the *ENV* variable, or implementation-defined system
32773 start-up files. Implementations may choose to disable the history list mechanism
32774 for users with appropriate privileges who do not set *HISTFILE*; the specific
32775 circumstances under which this occurs are implementation-defined. If more than
32776 one instance of the shell is using the same history file, it is unspecified how
32777 updates to the history file from those shells interact. As entries are deleted from
32778 the history file, they shall be deleted oldest first. It is unspecified when history file
32779 entries are physically removed from the history file. This volume of
32780 IEEE Std 1003.1-200x specifies the effects of this variable only for systems
32781 supporting the User Portability Utilities option.

32782 *HISTSIZE* Determine a decimal number representing the limit to the number of previous
32783 commands that are accessible. If this variable is unset, an unspecified default
32784 greater than or equal to 128 shall be used. The maximum number of commands in
32785 the history list is unspecified, but shall be at least 128. An implementation may
32786 choose to access this variable only when initializing the history file, as described
32787 under *HISTFILE*. Therefore, it is unspecified whether changes made to *HISTSIZE*
32788 after the history file has been initialized are effective.

32789 *HOME* Determine the pathname of the user's home directory. The contents of *HOME* are
32790 used in Tilde Expansion as described in Section 2.6.1 (on page 2239). This volume
32791 of IEEE Std 1003.1-200x specifies the effects of this variable only for systems
32792 supporting the User Portability Utilities option.

32793 *IFS* *Input field separators*: a string treated as a list of characters that shall be used for
32794 field splitting and to split lines into words with the *read* command. See Section
32795 2.6.5 (on page 2243). If *IFS* is not set, the shell shall behave as if the value of *IFS* |
32796 were <space>, <tab>, and <newline>. Implementations may ignore the value of |
32797 *IFS* in the environment at the time *sh* is invoked, treating *IFS* as if it were not set.

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the behavior of range expressions, equivalence classes and multi-character collating elements within pattern matching.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files), which characters are defined as letters (character class **alpha**), and the behavior of character classes within pattern matching.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

MAIL Determine 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 standard error prior to the writing of the next primary prompt string. Such check shall be performed only after the completion of the interval defined by the *MAILCHECK* variable after the last such check. The user shall be informed only if *MAIL* is set and *MAILPATH* is not set. This volume of IEEE Std 1003.1-200x specifies the effects of this variable only for systems supporting the User Portability Utilities option.

MAILCHECK
Establish 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 volume of IEEE Std 1003.1-200x specifies the effects of this variable only for systems supporting the User Portability Utilities option.

MAILPATH Provide 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.

The *MAILPATH* environment variable takes precedence over the *MAIL* variable. This volume of IEEE Std 1003.1-200x specifies the effects of this variable only for systems supporting the User Portability Utilities option.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

PATH Establish a string formatted as described in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables, used to effect command interpretation; see Section 2.9.1.1 (on page 2249).

PWD This variable shall represent an absolute pathname of the current working directory. Assignments to this variable may be ignored unless the value is an absolute pathname of the current working directory and there are no filename components of dot or dot-dot.

ASYNCHRONOUS EVENTS

Default.

STDOUT

See the STDERR section.

STDERR

Except as otherwise stated (by the descriptions of any invoked utilities or in interactive mode), standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

See Chapter 2. The following additional capabilities are supported on systems supporting the User Portability Utilities option.

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.

Command Line Editing

When *sh* is being used interactively from a terminal, the current command and the command history (see *fc* (on page 2637)) 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* 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 **Command Line Editing (vi-mode)** (on page 3053)). This command also shall disable any other editing mode that the implementation may provide. The command *set* **+o** *vi* disables *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 sections, the characters *erase*, *interrupt*, *kill*, and *end-of-file* are those set by the *stty* utility.

32882 **Command Line Editing (vi-mode)**

32883 In *vi* editing mode, there shall be a distinguished line, the edit line. All the editing operations
32884 which modify a line affect the edit line. The edit line is always the newest line in the command
32885 history buffer.

32886 With *vi*-mode enabled, *sh* can be switched between insert mode and command mode.

32887 When in insert mode, an entered character shall be inserted into the command line, except as
32888 noted in **vi Line Editing Insert Mode**. Upon entering *sh* and after termination of the previous
32889 command, *sh* shall be in insert mode.

32890 Typing an escape character shall switch *sh* into command mode (see **vi Line Editing Command**
32891 **Mode** (on page 3054)). In command mode, an entered character shall either invoke a defined
32892 operation, is used as part of a multi-character operation, or is treated as an error. A character that
32893 is not recognized as part of an editing command shall terminate any specific editing command
32894 and shall alert the terminal. Typing the *interrupt* character in command mode shall cause *sh* to
32895 terminate command line editing on the current command line, reissue the prompt on the next
32896 line of the terminal, and reset the command history (see *fc* (on page 2637)) so that the most
32897 recently executed command is the previous command (that is, the command that was being
32898 edited when it was interrupted is not reentered into the history).

32899 In the following sections, the phrase ''move the cursor to the beginning of the word'' shall mean
32900 ''move the cursor to the first character of the current word'' and the phrase ''move the cursor to
32901 the end of the word'' shall mean ''move the cursor to the last character of the current word''. The
32902 phrase ''beginning of the command line'' indicates the point between the end of the prompt
32903 string issued by the shell (or the beginning of the terminal line, if there is no prompt string) and
32904 the first character of the command text.

32905 **vi Line Editing Insert Mode**

32906 While in insert mode, any character typed shall be inserted in the current command line, unless
32907 it is from the following set.

32908 <newline> Execute the current command line. If the current command line is not empty, this
32909 line shall be entered into the command history (see *fc*).

32910 *erase* Delete the character previous to the current cursor position and move the current
32911 cursor position back one character. In insert mode, characters shall be erased from
32912 both the screen and the buffer when backspacing.

32913 *interrupt* Terminate command line editing with the same effects as described for
32914 interrupting command mode; see **Command Line Editing (vi-mode)**.

32915 *kill* Clear all the characters from the input line.

32916 <control>-V Insert the next character input, even if the character is otherwise a special insert
32917 mode character.

32918 <control>-W Delete the characters from the one preceding the cursor to the preceding word
32919 boundary. The word boundary in this case is the closer to the cursor of either the
32920 beginning of the line or a character that is in neither the **blank** nor **punct** character
32921 classification of the current locale.

32922 *end-of-file* Interpreted as the end of input in *sh*. This interpretation shall occur only at the
32923 beginning of an input line. If *end-of-file* is entered other than at the beginning of the
32924 line, the results are unspecified.

<ESC> Place *sh* into command mode.

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* represents one of the commands:

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

If the current line is not the edit line, any command that modifies the current line shall cause the content of the current line to replace the content of the edit line, and the current line shall become the edit line. This replacement cannot be undone (see the **u** and **U** commands below). The modification requested shall then be performed to the edit line. When the current line is the edit line, the modification shall be done directly to the edit line.

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. The term *save buffer* corresponds to the term *unnamed buffer* in *vi*.

The following commands shall be recognized in command mode:

<newline> Execute the current command line. If the current command line is not empty, this line shall be entered into the command history (see *fc*).

<control>-L Redraw the current command line. Position the cursor at the same location on the redrawn line.

Insert the character '#' at the beginning of the current command line and treat the resulting edit line as a comment. This line shall be entered into the command history; see *fc* (on page 2637).

= Display the possible shell word expansions (see Section 2.6 (on page 2238)) of the bigword at the current command line position.

Note: This does not modify the content of the current line, and therefore does not cause the current line to become the edit line.

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 is repositioned at the current cursor position, and *sh* shall be placed in command mode.

\ Perform pathname expansion (see Section 2.6.6 (on page 2244)) 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> 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>. 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 non-motion 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 and placed in the command history. If a *number* is included, 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. Any number argument on that previous command shall be ignored. 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 following 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 edit 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 characters 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 characters 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 (that is, copy) the characters from the current cursor position to the position resulting from the *motion* command into the 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 change the edit line. This operation shall not undo the copy of any command line to the edit line.

U Undo all changes made to the edit line. This operation shall not undo the copy of any command line to the edit line.

[*count***]k**
[*count***]–** Set the current command line to be the *count*th previous command line in the shell command history. If *count* is not specified, it shall default to 1. The cursor shall be positioned on the first character of the new command. If a **k** or – command would retreat 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***]+** Set the current command line to be the *count*th next command line in the shell command history. If *count* is not specified, it shall default to 1. The cursor shall be positioned on the first character of the new command. If a **j** or **+** command advances past the edit line, the current command line shall be restored to the edit line and terminal shall be alerted.

[*number***]G** Set the current command line to be the oldest command line stored in the shell command history. With a number *number*, set the current command line to be the command line *number* in the history. If command line *number* does not exist, the terminal shall be alerted and the command line shall not be changed.

/pattern<newline>
Move backwards through the command history, searching for the specified pattern, beginning with the previous command line. Patterns use the pattern matching notation described in Section 2.13 (on page 2264), except that the '^' character shall have special meaning when it appears as the first character of *pattern*. In this case, the '^' is discarded and the characters after the '^' shall be matched only at the beginning of a line. Commands in the command history shall be treated as strings, not as filenames. If the pattern is not found, the current command line shall be unchanged and the terminal is alerted. If it is found in a previous line, the current command line shall be set to that line and the cursor shall be set to the first character of the new command line.

If *pattern* is empty, the last non-empty pattern provided to **/** or **?** shall be used. If there is no previous non-empty pattern, the terminal shall be alerted and the current command line shall remain unchanged.

?*pattern*<newline>

Move forwards through the command history, searching for the specified pattern, beginning with the next command line. Patterns use the pattern matching notation described in Section 2.13 (on page 2264), except that the `'^'` character shall have special meaning when it appears as the first character of *pattern*. In this case, the `'^'` is discarded and the characters after the `'^'` shall be matched only at the beginning of a line. Commands in the command history shall be treated as strings, not as filenames. If the pattern is not found, the current command line shall be unchanged and the terminal alerted. If it is found in a following line, the current command line shall be set to that line and the cursor shall be set to the fist character of the new command line.

If *pattern* is empty, the last non-empty pattern provided to **/** or **?** shall be used. If there is no previous non-empty pattern, the terminal shall be alerted and the current command line shall remain unchanged.

n Repeat the most recent **/** or **?** command. If there is no previous **/** or **?**, the terminal shall be alerted and the current command line shall remain unchanged.

N Repeat the most recent **/** or **?** command, reversing the direction of the search. If there is no previous **/** or **?**, the terminal shall be alerted and the current command line shall remain unchanged.

EXIT STATUS

The following exit values shall be returned:

0 The script to be executed consisted solely of zero or more blank lines or comments, or both.

1-125 A non-interactive shell detected a syntax, redirection or variable assignment error.

127 A specified *command_file* could not be found by a non-interactive shell.

Otherwise, the shell shall return the exit status of the last command it invoked or attempted to invoke (see also the *exit* utility in Section 2.14 (on page 2266)).

CONSEQUENCES OF ERRORS

See Section 2.8.1 (on page 2247).

APPLICATION USAGE

Standard input and standard error are the files that determine whether a shell is interactive when **–i** is not specified. For example:

```
sh > file
```

and:

```
sh 2> file
```

create interactive and non-interactive shells, respectively. Although both accept terminal input, the results of error conditions are different, as described in Section 2.8.1 (on page 2247); in the second example a redirection error encountered by a special built-in utility aborts the shell.

A conforming application must protect its first operand, if it starts with a plus sign, by preceding it with the "−−" argument that denotes the end of the options.

Applications should note that the standard *PATH* to the shell cannot be assumed to be either **/bin/sh** or **/usr/bin/sh**, and should be determined by interrogation of the *PATH* returned by

getconf PATH, ensuring that the returned pathname is an absolute pathname and not a shell built in.

For example, to determine the location of the standard *sh* utility:

```
command –v sh
```

On some implementations this might return:

```
/usr/xpg4/bin/sh
```

Furthermore, on systems that support executable scripts (the "#!" construct), it is recommended that applications using executable scripts install them using *getconf* **–v** to determine the shell pathname and update the "#!" script appropriately as it is being installed (for example, with *sed*). For example:

```
#
# Installation time script to install correct POSIX shell pathname
#
# Get list of paths to check
#
Sifs=$IFS
IFS=:
set $(getconf PATH)
IFS=$Sifs
#
# Check each path for 'sh'
#
for i in $@
do
    if [ –f ${i}/sh ];
    then
        Pshell=${i}/sh
    fi
done
#
# This is the list of scripts to update. They should be of the
# form '${name}.source' and will be transformed to '${name}'.
# Each script should begin:
#
# !INSTALLSHELLPATH –p
#
scripts="a b c"
#
# Transform each script
#
for i in ${scripts}
do
    sed –e "s|INSTALLSHELLPATH|${Pshell}|" < ${i}.source > ${i}
done
```

EXAMPLES

1. Execute a shell command from a string:

   ```
   sh –c "cat myfile"
   ```

2. Execute a shell script from a file in the current directory:

```
sh my_shell_cmds
```

RATIONALE

The *sh* utility and the *set* special built-in utility share a common set of options.

The KornShell ignores the contents of *IFS* upon entry to the script. A conforming application cannot rely on importing *IFS*. One justification for this, beyond security considerations, is to assist possible future shell compilers. Allowing *IFS* to be imported from the environment prevents many optimizations that might otherwise be performed via dataflow analysis of the script itself.

The text in the STDIN section about non-blocking reads concerns an instance of *sh* that has been invoked, probably by a C-language program, with standard input that has been opened using the O_NONBLOCK flag; see *open*( ) in the System Interfaces volume of IEEE Std 1003.1-200x. If the shell did not reset this flag, it would immediately terminate because no input data would be available yet and that would be considered the same as end-of-file.

The options associated with a *restricted shell* (command name *rsh* and the **–r** option) were excluded because the standard developers considered that the implied level of security could not be achieved and they did not want to raise false expectations.

On systems that support set-user-ID scripts, a historical trapdoor has been to link a script to the name **–i**. When it is called by a sequence such as

```
sh –
```

or by:

```
#! usr/bin/sh –
```

the historical systems have assumed that no option letters follow. Thus, this volume of IEEE Std 1003.1-200x allows the single hyphen to mark the end of the options, in addition to the use of the regular "––" argument, because it was considered that the older practice was so pervasive. An alternative approach is taken by the KornShell, where real and effective user/group IDs must match for an interactive shell; this behavior is specifically allowed by this volume of IEEE Std 1003.1-200x.

Note: There are other problems with set-user-ID scripts that the two approaches described here do not resolve.

The initialization process for the history file can be dependent on the system start-up files, in that they may contain commands that effectively preempt the user's settings of *HISTFILE* and *HISTSIZE*. For example, function definition commands are recorded in the history file, unless the *set* **–o** *nolog* option is set. If the system administrator includes function definitions in some system start-up file called before the *ENV* file, the history file is initialized before the user gets a chance to influence its characteristics.) In some historical shells, the history file is initialized just after the *ENV* file has been processed. Therefore, it is implementation-defined whether changes made to *HISTFILE* after the history file has been initialized are effective.

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 IEEE Std 1003.1-200x 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 early proposals, the KornShell-derived *emacs* mode of command line editing was included, even though the *emacs* editor itself was not. The community of *emacs* proponents was adamant that the full *emacs* editor not be included in earlier versions of IEEE Std 1003.1 because they were concerned that an attempt to standardize this very powerful environment would encourage vendors to ship versions conforming strictly to earlier versions of IEEE Std 1003.1, 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 earlier versions of IEEE Std 1003.1 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 standard developers decided to defer specification at this time, rather than attempting to agree on a specific subset of *emacs* late within the development of earlier versions of IEEE Std 1003.1. At the time, it was assumed that convergence on an acceptable definition would occur for a subsequent draft, but that has not happened, and there appears to be no impetus to do so. In any case, implementations are free to offer additional command line editing modes based on the exact models of editors their users are most comfortable with.

Early proposals had the following list entry in **vi Line Editing Insert Mode** (on page 3053):

\ 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.

However, this is not actually a feature of *sh* command line editing insert mode, but one of some historical terminal line drivers. Some conforming implementations continue to do this when the *stty* **iexten** flag is set.

FUTURE DIRECTIONS

None.

SEE ALSO

cd, *echo*, *pwd*, *test*, *umask*, the System Interfaces volume of IEEE Std 1003.1-200x, *dup*( ), *exec*, *exit*( ), *fork*( ), *pipe*( ), *signal*( ), *system*( ), *ulimit*( ), *umask*( ), *wait*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

FUTURE DIRECTIONS section added.

Text is added to the DESCRIPTION for the Large File Summit proposal.

Issue 6

The Open Group Corrigendum U029/2 is applied, correcting the second SYNOPSIS.

The Open Group Corrigendum U027/3 is applied, correcting a typographical error.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The option letters derived from the set special built-in are also accepted with a leading plus sign ('+').
- Large file extensions are added:
  - — Pathname expansion does not fail due to the size of a file.
  - — Shell input and output redirections have an implementation-defined offset maximum that is established in the open file description.

In the ENVIRONMENT VARIABLES section, the text ''user's home directory'' is updated to ''directory referred to by the *HOME* environment variable''.

Descriptions for the *ENV* and *PWD* environment variables are included to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ''must'' for application requirements.

NAME

sleep — suspend execution for an interval

SYNOPSIS

```
sleep time
```

DESCRIPTION

The *sleep* utility shall suspend execution for at least the integral number of seconds specified by the *time* operand.

OPTIONS

None.

OPERANDS

The following operand shall be supported:

time A non-negative decimal integer specifying the number of seconds for which to suspend execution.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *sleep*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

If the *sleep* utility receives a SIGALRM signal, one of the following actions shall be taken:

1. Terminate normally with a zero exit status.
2. Effectively ignore the signal.
3. Provide the default behavior for signals described in the ASYNCHRONOUS EVENTS section of Section 1.11 (on page 2221). This could include terminating with a non-zero exit status.

The *sleep* utility shall take the standard action for all other signals.

33441 **STDOUT**
33442 Not used.

33443 **STDERR**
33444 The standard error shall be used only for diagnostic messages.

33445 **OUTPUT FILES**
33446 None.

33447 **EXTENDED DESCRIPTION**
33448 None.

33449 **EXIT STATUS**
33450 The following exit values shall be returned:

33451 0 The execution was successfully suspended for at least *time* seconds, or a SIGALRM signal
33452 was received. See the ASYNCHRONOUS EVENTS section.

33453 >0 An error occurred.

33454 **CONSEQUENCES OF ERRORS**
33455 Default.

33456 **APPLICATION USAGE**
33457 None.

33458 **EXAMPLES**
33459 The *sleep* utility can be used to execute a command after a certain amount of time, as in:

33460 `(sleep 105; command) &`

33461 or to execute a command every so often, as in:

```
33462 while true
33463 do
33464     command
33465     sleep 37
33466 done
```

33467 **RATIONALE**
33468 The exit status is allowed to be zero when *sleep* is interrupted by the SIGALRM signal because
33469 most implementations of this utility rely on the arrival of that signal to notify them that the
33470 requested finishing time has been successfully attained. Such implementations thus do not
33471 distinguish this situation from the successful completion case. Other implementations are
33472 allowed to catch the signal and go back to sleep until the requested time expires or to provide
33473 the normal signal termination procedures.

33474 As with all other utilities that take integral operands and do not specify subranges of allowed
33475 values, *sleep* is required by this volume of IEEE Std 1003.1-200x to deal with *time* requests of up
33476 to 2 147 483 647 seconds. This may mean that some implementations have to make multiple calls
33477 to the delay mechanism of the underlying operating system if its argument range is less than
33478 this.

33479 **FUTURE DIRECTIONS**
33480 None.

33481 **SEE ALSO**
33482 *wait*, the System Interfaces volume of IEEE Std 1003.1-200x, *alarm*( ), *sleep*( )

33483 **CHANGE HISTORY**
33484 First released in Issue 2.

33485 **NAME**

33486 sort — sort, merge, or sequence check text files

33487 **SYNOPSIS**

33488 `sort [–m][–o output][–bdfinru][–t char][–k keydef]... [file...]`

33489 `sort –c [–bdfinru][–t char][–k keydef][file]`

33490 **DESCRIPTION**

33491 The *sort* utility shall perform one of the following functions:

33492 1. Sort lines of all the named files together and write the result to the specified output.

33493 2. Merge lines of all the named (presorted) files together and write the result to the specified
33494 output.

33495 3. Check that a single input file is correctly presorted.

33496 Comparisons shall be based on one or more sort keys extracted from each line of input (or, if no
33497 sort keys are specified, the entire line up to, but not including, the terminating <newline>), and
33498 shall be performed using the collating sequence of the current locale.

33499 **OPTIONS**

33500 The *sort* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
33501 12.2, Utility Syntax Guidelines, and the **–k** *keydef* option should follow the **–b**, **–d**, **–f**, **–i**, **–n**, and
33502 **–r** options.

33503 The following options shall be supported:

33504 **–c** Check that the single input file is ordered as specified by the arguments and the
33505 collating sequence of the current locale. No output shall be produced; only the exit
33506 code shall be affected.

33507 **–m** Merge only; the input file shall be assumed to be already sorted.

33508 **–o** *output* Specify the name of an output file to be used instead of the standard output. This
33509 file can be the same as one of the input *file*s.

33510 **–u** Unique: suppress all but one in each set of lines having equal keys. If used with
33511 the **–c** option, check that there are no lines with duplicate keys, in addition to
33512 checking that the input file is sorted.

33513 The following options shall override the default ordering rules. When ordering options appear
33514 independent of any key field specifications, the requested field ordering rules shall be applied
33515 globally to all sort keys. When attached to a specific key (see **–k**), the specified ordering options
33516 shall override all global ordering options for that key.

33517 **–d** Specify that only <blank>s and alphanumeric characters, according to the current
33518 setting of *LC_CTYPE*, shall be significant in comparisons. The behavior is
33519 undefined for a sort key to which **–i** or **–n** also applies.

33520 **–f** Consider all lowercase characters that have uppercase equivalents, according to
33521 the current setting of *LC_CTYPE*, to be the uppercase equivalent for the purposes
33522 of comparison.

33523 **–i** Ignore all characters that are non-printable, according to the current setting of
33524 *LC_CTYPE*.

33525 **–n** Restrict the sort key to an initial numeric string, consisting of optional <blank>s,
33526 optional minus sign, and zero or more digits with an optional radix character and
33527 thousands separators (as defined in the current locale), which shall be sorted by

arithmetic value. An empty digit string shall be treated as zero. Leading zeros and signs on zeros shall not affect ordering.

–r Reverse the sense of comparisons.

The treatment of field separators can be altered using the options:

–b Ignore leading <blank>s when determining the starting and ending positions of a restricted sort key. If the **–b** option is specified before the first **–k** option, it shall be applied to all **–k** options. Otherwise, the **–b** option can be attached independently to each **–k** *field_start* or *field_end* option-argument (see below).

–t *char* Use *char* as the field separator character; *char* shall not be considered to be part of a field (although it can be included in a sort key). Each occurrence of *char* shall be significant (for example, <*char*><*char*> delimits an empty field). If **–t** is not specified, <blank>s shall be used as default field separators; each maximal non-empty sequence of <blank>s that follows a non-<blank> shall be a field separator.

Sort keys can be specified using the options:

–k *keydef* The *keydef* argument is a restricted sort key field definition. The format of this definition is:

field_start`[`*type*`][,`*field_end*`[`*type*`]]`

where *field_start* and *field_end* define a key field restricted to a portion of the line (see the EXTENDED DESCRIPTION section), and *type* is a modifier from the list of characters `'b'`, `'d'`, `'f'`, `'i'`, `'n'`, `'r'`. The `'b'` modifier shall behave like the **–b** option, but shall apply only to the *field_start* or *field_end* to which it is attached. The other modifiers shall behave like the corresponding options, but shall apply only to the key field to which they are attached; they shall have this effect if specified with *field_start*, *field_end*, or both. If any modifier is attached to a *field_start* or to a *field_end*, no option shall apply to either. Implementations shall support at least nine occurrences of the **–k** option, which shall be significant in command line order. If no **–k** option is specified, a default sort key of the entire line shall be used.

When there are multiple key fields, later keys shall be compared only after all earlier keys compare equal. Except when the **–u** option is specified, lines that otherwise compare equal shall be ordered as if none of the options **–d**, **–f**, **–i**, **–n**, or **–k** were present (but with **–r** still in effect, if it was specified) and with all bytes in the lines significant to the comparison. The order in which lines that still compare equal are written is unspecified.

OPERANDS

The following operand shall be supported:

file A pathname of a file to be sorted, merged, or checked. If no *file* operands are specified, or if a *file* operand is `'–'`, the standard input shall be used.

STDIN

The standard input shall be used only if no *file* operands are specified, or if a *file* operand is `'–'`. See the INPUT FILES section.

INPUT FILES

The input files shall be text files, except that the *sort* utility shall add a <newline> to the end of a file ending with an incomplete last line.

33572 **ENVIRONMENT VARIABLES**

33573 The following environment variables shall affect the execution of *sort*:

33574 *LANG* Provide a default value for the internationalization variables that are unset or null.
33575 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
33576 Internationalization Variables for the precedence of internationalization variables
33577 used to determine the values of locale categories.)

33578 *LC_ALL* If set to a non-empty string value, override the values of all the other
33579 internationalization variables.

33580 *LC_COLLATE*
33581 Determine the locale for ordering rules.

33582 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
33583 characters (for example, single-byte as opposed to multi-byte characters in
33584 arguments and input files) and the behavior of character classification for the **–b**,
33585 **–d**, **–f**, **–i**, and **–n** options.

33586 *LC_MESSAGES*
33587 Determine the locale that should be used to affect the format and contents of
33588 diagnostic messages written to standard error.

33589 *LC_NUMERIC*
33590 Determine the locale for the definition of the radix character and thousands
33591 separator for the **–n** option.

33592 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

33593 **ASYNCHRONOUS EVENTS**

33594 Default.

33595 **STDOUT**

33596 Unless the **–o** or **–c** options are in effect, the standard output shall contain the sorted input.

33597 **STDERR**

33598 The standard error shall be used for diagnostic messages. A warning message about correcting |
33599 an incomplete last line of an input file may be generated, but need not affect the final exit status. |

33600 **OUTPUT FILES**

33601 If the **–o** option is in effect, the sorted input shall be written to the file *output*.

33602 **EXTENDED DESCRIPTION**

33603 The notation:

33604 –k *field_start*[*type*][,*field_end*[*type*]]

33605 shall define a key field that begins at *field_start* and ends at *field_end* inclusive, unless *field_start*
33606 falls beyond the end of the line or after *field_end*, in which case the key field is empty. A missing
33607 *field_end* shall mean the last character of the line.

33608 A field comprises a maximal sequence of non-separating characters and, in the absence of option
33609 **–t**, any preceding field separator.

33610 The *field_start* portion of the *keydef* option-argument shall have the form:

33611 *field_number*[.*first_character*]

33612 Fields and characters within fields shall be numbered starting with 1. The *field_number* and
33613 *first_character* pieces, interpreted as positive decimal integers, shall specify the first character to
33614 be used as part of a sort key. If .*first_character* is omitted, it shall refer to the first character of the

field.

The *field_end* portion of the *keydef* option-argument shall have the form:

field_number[.*last_character*]

The *field_number* shall be as described above for *field_start*. The *last_character* piece, interpreted as a non-negative decimal integer, shall specify the last character to be used as part of the sort key. If *last_character* evaluates to zero or .*last_character* is omitted, it shall refer to the last character of the field specified by *field_number*.

If the **–b** option or **b** type modifier is in effect, characters within a field shall be counted from the first non-<blank> in the field. (This shall apply separately to *first_character* and *last_character*.)

EXIT STATUS

The following exit values shall be returned:

0 All input files were output successfully, or **–c** was specified and the input file was correctly sorted.

1 Under the **–c** option, the file was not ordered as specified, or if the **–c** and **–u** options were both specified, two input lines were found with equal keys.

>1 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The default value for **–t**, <blank>, has different properties from, for example, **–t**"<space>". If a line contains:

```
<space><space>foo
```

the following treatment would occur with default separation as opposed to specifically selecting a <space>:

Field	Default	–t "<space>"
1	<space><space>foo	*empty*
2	*empty*	*empty*
3	*empty*	foo

The leading field separator itself is included in a field when **–t** is not used. For example, this command returns an exit status of zero, meaning the input was already sorted:

```
sort –c –k 2 <<eof
y<tab>b
x<space>a
eof
```

(assuming that a <tab> precedes the <space> in the current collating sequence). The field separator is not included in a field when it is explicitly set via **–t**. This is historical practice and allows usage such as:

```
sort –t "|" –k 2n <<eof
Atlanta|425022|Georgia
Birmingham|284413|Alabama
Columbia|100385|South Carolina
eof
```

where the second field can be correctly sorted numerically without regard to the non-numeric field separator.

The wording in the OPTIONS section clarifies that the –**b**, –**d**, –**f**, –**i**, –**n**, and –**r** options have to come before the first sort key specified if they are intended to apply to all specified keys. The way it is described in this volume of IEEE Std 1003.1-200x matches historical practice, not historical documentation. The results are unspecified if these options are specified after a –**k** option.

The –**f** option might not work as expected in locales where there is not a one-to-one mapping between an uppercase and a lowercase letter.

EXAMPLES

1. The following command sorts the contents of **infile** with the second field as the sort key:

   ```
   sort –k 2,2 infile
   ```

2. The following command sorts, in reverse order, the contents of **infile1** and **infile2**, placing the output in **outfile** and using the second character of the second field as the sort key (assuming that the first character of the second field is the field separator):

   ```
   sort –r –o outfile –k 2.2,2.2 infile1 infile2
   ```

3. The following command sorts the contents of **infile1** and **infile2** using the second non-<blank> of the second field as the sort key:

   ```
   sort –k 2.2b,2.2b infile1 infile2
   ```

4. The following command prints the System V password file (user database) sorted by the numeric user ID (the third colon-separated field):

   ```
   sort –t : –k 3,3n /etc/passwd
   ```

5. The following command prints the lines of the already sorted file **infile**, suppressing all but one occurrence of lines having the same third field:

   ```
   sort –um –k 3.1,3.0 infile
   ```

RATIONALE

Examples in some historical documentation state that options –**um** with one input file keep the first in each set of lines with equal keys. This behavior was deemed to be an implementation artifact and was not standardized.

The –**z** option was omitted; it is not standard practice on most systems and is inconsistent with using *sort* to sort several files individually and then merge them together. The text concerning –**z** in historical documentation appeared to require implementations to determine the proper buffer length during the sort phase of operation, but not during the merge.

The –**y** option was omitted because of non-portability. The –**M** option, present in System V, was omitted because of non-portability in international usage.

An undocumented –**T** option exists in some implementations. It is used to specify a directory for intermediate files. Implementations are encouraged to support the use of the *TMPDIR* environment variable instead of adding an option to support this functionality.

The –**k** option was added to satisfy two objections. First, the zero-based counting used by *sort* is not consistent with other utility conventions. Second, it did not meet syntax guideline requirements.

Historical documentation indicates that ‘‘setting –**n** implies –**b**’’. The description of –**n** already states that optional leading <blank>s are tolerated in doing the comparison. If –**b** is enabled,

rather than implied, by –**n**, this has unusual side effects. When a character offset is used in a column of numbers (for example, to sort modulo 100), that offset is measured relative to the most significant digit, not to the column. Based upon a recommendation from the author of the original *sort* utility, the –**b** implication has been omitted from this volume of IEEE Std 1003.1-200x, and an application wishing to achieve the previously mentioned side effects has to code the –**b** flag explicitly.

FUTURE DIRECTIONS

None.

SEE ALSO

comm, *join*, *uniq*, the System Interfaces volume of IEEE Std 1003.1-200x, *toupper*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

IEEE PASC Interpretation 1003.2 #174 is applied, updating the DESCRIPTION of comparisons.

IEEE PASC Interpretation 1003.2 #168 is applied.

NAME

split — split files into pieces

SYNOPSIS

UP

```
split [–l line_count][–a suffix_length][file[name]]
split –b n[k|m][–a suffix_length][file[name]]
```

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 1 000 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'`, and so on, until a name of all `'z'` characters 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"`, and so on, 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* shall not delete the files it created with valid suffixes. If the file limit is not exceeded, the last file created shall contain the remainder of the input file, and may be smaller than the requested size.

OPTIONS

The *split* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–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*
: Specify the number of lines in each resulting file piece. The *line_count* argument is an unsigned decimal integer. The default is 1 000. If the input does not end with a <newline>, the partial line shall be included in the last output file.

OPERANDS

The following operands shall be supported:

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 the OPTIONS section.

STDIN

See the INPUT FILES section.

INPUT FILES

Any file can be used as input.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *split*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

The output files contain portions of the original input file; otherwise, unchanged.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

33795 **APPLICATION USAGE**

33796 None.

33797 **EXAMPLES**

33798 In the following examples **foo** is a text file that contains 5 000 lines.

33799 1. Create five files, **xaa**, **xab**, **xac**, **xad**, and **xae**:

33800 `split foo`

33801 2. Create five files, but the suffixed portion of the created files consists of three letters, **xaaa**,
33802 **xaab**, **xaac**, **xaad**, and **xaae**:

33803 `split –a 3 foo`

33804 3. Create three files with four-letter suffixes and a supplied prefix, **bar_aaaa**, **bar_aaab**, and
33805 **bar_aaac**:

33806 `split –a 4 –l 2000 foo bar_`

33807 4. Create as many files as are necessary to contain at most 20*1 024 bytes, each with the
33808 default prefix of **x** and a five-letter suffix:

33809 `split –a 5 –b 20k foo`

33810 **RATIONALE**

33811 The **–b** option was added to provide a mechanism for splitting files other than by lines. While
33812 most uses of the **–b** option are for transmitting files over networks, some believed it would have
33813 additional uses.

33814 The **–a** option was added to overcome the limitation of being able to create only 676 files.

33815 Consideration was given to deleting this utility, using the rationale that the function provided
33816 by this utility is available via the *csplit* utility (see *csplit* (on page 2480)). Upon reconsideration of
33817 the purpose of the User Portability Extension, it was decided to retain both this utility and the
33818 *csplit* utility because users use both utilities and have historical expectations of their behavior.
33819 Furthermore, the splitting on byte boundaries in *split* cannot be duplicated with the historical
33820 *csplit*.

33821 The text ‘‘*split* shall not delete the files it created with valid suffixes’’ would normally be
33822 assumed, but since the related utility, *csplit*, does delete files under some circumstances, the
33823 historical behavior of *split* is made explicit to avoid misinterpretation.

33824 **FUTURE DIRECTIONS**

33825 None.

33826 **SEE ALSO**

33827 *csplit*

33828 **CHANGE HISTORY**

33829 First released in Issue 2.

33830 **Issue 6**

33831 This utility is now marked as part of the User Portability Utilities option.

33832 The APPLICATION USAGE section is added.

33833 The obsolescent SYNOPSIS is removed.

33834 **NAME**

33835 strings — find printable strings in files

33836 **SYNOPSIS**

33837 UP

```
strings [–a][–t format][–n number][file...]
```

33838

33839 **DESCRIPTION**

33840 The *strings* utility shall look for printable strings in regular files and shall write those strings to
33841 standard output. A printable string is any sequence of four (by default) or more printable
33842 characters terminated by a <newline> or NUL character. Additional implementation-defined
33843 strings may be written; see *localedef*.

33844 **OPTIONS**

33845 The *strings* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
33846 12.2, Utility Syntax Guidelines.

33847 The following options shall be supported:

33848 **–a** Scan files in their entirety. If **–a** is not specified, it is implementation-defined what
33849 portion of each file is scanned for strings.

33850 **–n** *number* Specify the minimum string length, where the *number* argument is a positive
33851 decimal integer. The default shall be 4.

33852 **–t** *format* Write each string preceded by its byte offset from the start of the file. The format
33853 shall be dependent on the single character used as the *format* option-argument:

33854 d The offset shall be written in decimal.

33855 o The offset shall be written in octal.

33856 x The offset shall be written in hexadecimal.

33857 **OPERANDS**

33858 The following operand shall be supported:

33859 *file* A pathname of a regular file to be used as input. If no *file* operand is specified, the
33860 *strings* utility shall read from the standard input.

33861 **STDIN**

33862 See the INPUT FILES section.

33863 **INPUT FILES**

33864 The input files named by the utility arguments or the standard input shall be regular files of any
33865 format.

33866 **ENVIRONMENT VARIABLES**

33867 The following environment variables shall affect the execution of *strings*:

33868 *LANG* Provide a default value for the internationalization variables that are unset or null.
33869 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
33870 Internationalization Variables for the precedence of internationalization variables
33871 used to determine the values of locale categories.)

33872 *LC_ALL* If set to a non-empty string value, override the values of all the other
33873 internationalization variables.

33874 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
33875 characters (for example, single-byte as opposed to multi-byte characters in
33876 arguments and input files) and to identify printable strings.

33877 *LC_MESSAGES*
33878 Determine the locale that should be used to affect the format and contents of
33879 diagnostic messages written to standard error.

33880 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

33881 **ASYNCHRONOUS EVENTS**
33882 Default.

33883 **STDOUT**
33884 Strings found shall be written to the standard output, one per line.

33885 When the **–t** option is not specified, the format of the output shall be:

33886 `"%s", <string>`

33887 With the **–t o** option, the format of the output shall be:

33888 `"%o %s", <byte offset>, <string>`

33889 With the **–t x** option, the format of the output shall be:

33890 `"%x %s", <byte offset>, <string>`

33891 With the **–t d** option, the format of the output shall be:

33892 `"%d %s", <byte offset>, <string>`

33893 **STDERR**
33894 The standard error shall be used only for diagnostic messages.

33895 **OUTPUT FILES**
33896 None.

33897 **EXTENDED DESCRIPTION**
33898 None.

33899 **EXIT STATUS**
33900 The following exit values shall be returned:

33901 0 Successful completion.

33902 >0 An error occurred.

33903 **CONSEQUENCES OF ERRORS**
33904 Default.

33905 **APPLICATION USAGE**
33906 By default the data area (as opposed to the text, ‘‘bss’’ or header areas) of a binary executable file
33907 is scanned. Implementations document which areas are scanned.

33908 Some historical implementations do not require NUL or <newline> terminators for strings to
33909 permit those languages that do not use NUL as a string terminator to have their strings written.

33910 **EXAMPLES**
33911 None.

33912 **RATIONALE**
33913 Apart from rationalizing the option syntax and slight difficulties with object and executable
33914 binary files, *strings* is specified to match historical practice closely. The **–a** and **–n** options were
33915 introduced to replace the non-conforming – and *–number* options.

33916 The **–o** option historically means different things on different implementations. Some use it to
33917 mean ‘‘*offset* in decimal’’, while others use it as ‘‘*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*.

The ISO C standard function *isprint*( ) is restricted to a domain of **unsigned char**. This volume of IEEE Std 1003.1-200x requires implementations to write strings as defined by the current locale.

FUTURE DIRECTIONS

None.

SEE ALSO

nm

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The obsolescent SYNOPSIS is removed.

The normative text is reworded to avoid use of the term ''must'' for application requirements.

33932 **NAME**

33933 strip — remove unnecessary information from executable files (**DEVELOPMENT**)

33934 **SYNOPSIS**

33935 SD
```
strip file...
```
33936

33937 **DESCRIPTION**

33938 The *strip* utility shall remove from executable files named by the *file* operands any information
33939 the implementor deems unnecessary for execution of those files. The nature of that information
33940 is unspecified. The effect of *strip* shall be similar to the use of the **–s** option to *c99* or *fort77*.

33941 **OPTIONS**

33942 None.

33943 **OPERANDS**

33944 The following operand shall be supported:

33945 *file* A pathname referring to an executable file.

33946 **STDIN**

33947 Not used.

33948 **INPUT FILES**

33949 The input files shall be in the form of executable files successfully produced by any compiler
33950 defined by this volume of IEEE Std 1003.1-200x.

33951 **ENVIRONMENT VARIABLES**

33952 The following environment variables shall affect the execution of *strip*:

33953 *LANG* Provide a default value for the internationalization variables that are unset or null.
33954 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
33955 Internationalization Variables for the precedence of internationalization variables
33956 used to determine the values of locale categories.)

33957 *LC_ALL* If set to a non-empty string value, override the values of all the other
33958 internationalization variables.

33959 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
33960 characters (for example, single-byte as opposed to multi-byte characters in
33961 arguments).

33962 *LC_MESSAGES*
33963 Determine the locale that should be used to affect the format and contents of
33964 diagnostic messages written to standard error.

33965 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

33966 **ASYNCHRONOUS EVENTS**

33967 Default.

33968 **STDOUT**

33969 Not used.

33970 **STDERR**

33971 The standard error shall be used only for diagnostic messages.

OUTPUT FILES

The *strip* utility shall produce executable files of unspecified format.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

Historically, this utility has been used to remove the symbol table from an executable file. It was included since it is known that the amount of symbolic information can amount to several megabytes; the ability to remove it in a portable manner was deemed important, especially for smaller systems.

The behavior of *strip* is said to be the same as the **–s** option to a compiler. While the end result is essentially the same, it is not required to be identical.

FUTURE DIRECTIONS

None.

SEE ALSO

ar, *c99*, *fort77*

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is now marked as part of the Software Development Utilities option.

34001 **NAME**

34002 stty — set the options for a terminal

34003 **SYNOPSIS**

34004 `stty [ –a| –g]`

34005 `stty` *operands*

34006 **DESCRIPTION**

34007 The *stty* utility shall set or report on terminal I/O characteristics for the device that is its
34008 standard input. Without options or operands specified, it shall report the settings of certain
34009 characteristics, usually those that differ from implementation-defined defaults. Otherwise, it
34010 shall modify the terminal state according to the specified operands. Detailed information about
34011 the modes listed in the first five groups below are described in the Base Definitions volume of
34012 IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface. Operands in the Combination
34013 Modes group (see **Combination Modes** (on page 3087)) are implemented using operands in the
34014 previous groups. Some combinations of operands are mutually-exclusive on some terminal
34015 types; the results of using such combinations are unspecified.

34016 Typical implementations of this utility require a communications line configured to use the
34017 **termios** interface defined in the System Interfaces volume of IEEE Std 1003.1-200x. On systems
34018 where none of these lines are available, and on lines not currently configured to support the
34019 **termios** interface, some of the operands need not affect terminal characteristics.

34020 **OPTIONS**

34021 The *stty* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
34022 12.2, Utility Syntax Guidelines.

34023 The following options shall be supported:

34024 **–a** Write to standard output all the current settings for the terminal.

34025 **–g** Write to standard output all the current settings in an unspecified form that can be
34026 used as arguments to another invocation of the *stty* utility on the same system. The
34027 form used shall not contain any characters that would require quoting to avoid
34028 word expansion by the shell; see Section 2.6 (on page 2238).

34029 **OPERANDS**

34030 The following operands shall be supported to set the terminal characteristics.

34031 **Control Modes**

34032 **parenb** (**–parenb**) Enable (disable) parity generation and detection. This shall have the effect of |
34033 setting (not setting) PARENB in the **termios** *c_cflag* field, as defined in the |
34034 Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General
34035 Terminal Interface.

34036 **parodd** (**–parodd**) Select odd (even) parity. This shall have the effect of setting (not setting)
34037 PARODD in the **termios** *c_cflag* field, as defined in the Base Definitions
34038 volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

34039 **cs5 cs6 cs7 cs8** Select character size, if possible. This shall have the effect of setting CS5, CS6,
34040 CS7, and CS8, respectively, in the **termios** *c_cflag* field, as defined in the Base
34041 Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal
34042 Interface.

34043 *number* Set terminal baud rate to the number given, if possible. If the baud rate is set
34044 to zero, the modem control lines shall not be longer asserted. This shall have
34045 the effect of setting the input and output **termios** baud rate values as defined

in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

ispeed *number* Set terminal input baud rate to the number given, if possible. If the input baud rate is set to zero, the input baud rate shall be specified by the value of the output baud rate. This shall have the effect of setting the input **termios** baud rate values as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

ospeed *number* Set terminal output baud rate to the number given, if possible. If the output baud rate is set to zero, the modem control lines shall no longer be asserted. This shall have the effect of setting the output **termios** baud rate values as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

hupcl (**–hupcl**) Stop asserting modem control lines (do not stop asserting modem control lines) on last close. This shall have the effect of setting (not setting) HUPCL in the **termios** *c_cflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

hup (**–hup**) Equivalent to **hupcl**(**–hupcl**).

cstopb (**–cstopb**) Use two (one) stop bits per character. This shall have the effect of setting (not setting) CSTOPB in the **termios** *c_cflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

cread (**–cread**) Enable (disable) the receiver. This shall have the effect of setting (not setting) CREAD in the **termios** *c_cflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

clocal (**–clocal**) Assume a line without (with) modem control. This shall have the effect of setting (not setting) CLOCAL in the **termios** *c_cflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

It is unspecified whether *stty* shall report an error if an attempt to set a Control Mode fails.

Input Modes

ignbrk (**–ignbrk**) Ignore (do not ignore) break on input. This shall have the effect of setting (not setting) IGNBRK in the **termios** *c_iflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

brkint (**–brkint**) Signal (do not signal) INTR on break. This shall have the effect of setting (not setting) BRKINT in the **termios** *c_iflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

ignpar (**–ignpar**) Ignore (do not ignore) bytes with parity errors. This shall have the effect of setting (not setting) IGNPAR in the **termios** *c_iflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

parmrk (**–parmrk**)
Mark (do not mark) parity errors. This shall have the effect of setting (not setting) PARMRK in the **termios** *c_iflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

inpck (**–inpck**) Enable (disable) input parity checking. This shall have the effect of setting (not setting) INPCK in the **termios** *c_iflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

istrip (**–istrip**) Strip (do not strip) input characters to seven bits. This shall have the effect of setting (not setting) ISTRIP in the **termios** *c_iflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

inlcr (**–inlcr**) Map (do not map) NL to CR on input. This shall have the effect of setting (not setting) INLCR in the **termios** *c_iflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

igncr (–igncr) Ignore (do not ignore) CR on input. This shall have the effect of setting (not setting) IGNCR in the **termios** *c_iflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

icrnl (**–icrnl**) Map (do not map) CR to NL on input. This shall have the effect of setting (not setting) ICRNL in the **termios** *c_iflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

ixon (**–ixon**) Enable (disable) START/STOP output control. Output from the system is stopped when the system receives STOP and started when the system receives START. This shall have the effect of setting (not setting) IXON in the **termios** *c_iflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

XSI **ixany** (**–ixany**) Allow any character to restart output. This shall have the effect of setting (not setting) IXANY in the **termios** *c_iflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

ixoff (**–ixoff**) Request that the system send (not send) STOP characters when the input queue is nearly full and START characters to resume data transmission. This shall have the effect of setting (not setting) IXOFF in the **termios** *c_iflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

Output Modes

opost (**–opost**) Post-process output (do not post-process output; ignore all other output modes). This shall have the effect of setting (not setting) OPOST in the **termios** *c_oflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

XSI **ocrnl** (**–ocrnl**) Map (do not map) CR to NL on output This shall have the effect of setting (not setting) OCRNL in the **termios** *c_oflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

onocr (**–onocr**) Do not (do) output CR at column zero. This shall have the effect of setting (not setting) ONOCR in the **termios** *c_oflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

onlret (**–onlret**) The terminal newline key performs (does not perform) the CR function. This shall have the effect of setting (not setting) ONLRET in the **termios** *c_oflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

ofill (**–ofill**) Use fill characters (use timing) for delays. This shall have the effect of setting (not setting) OFILL in the **termios** *c_oflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

ofdel (**–ofdel**) Fill characters are DELs (NULs). This shall have the effect of setting (not setting) OFDEL in the **termios** *c_oflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

cr0 cr1 cr2 cr3 Select the style of delay for CRs. This shall have the effect of setting CRDLY to CR0, CR1, CR2, or CR3, respectively, in the **termios** *c_oflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

nl0 nl1 Select the style of delay for NL. This has the effect of setting NLDLY to NL0 or NL1, respectively, in the **termios** *c_oflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

tab0 tab1 tab2 tab3
Select the style of delay for horizontal tabs. This shall have the effect of setting TABDLY to TAB0, TAB1, TAB2, or TAB3, respectively, in the **termios** *c_oflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface. Note that TAB3 has the effect of expanding <tab>s to <space>s.

tabs (**–tabs**) Synonym for **tab0** (**tab3**).

bs0 bs1 Select the style of delay for backspaces. This shall have the effect of setting BSDLY to BS0 or BS1, respectively, in the **termios** *c_oflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

ff0 ff1 Select the style of delay for form-feeds. This shall have the effect of setting FFDLY to FF0 or FF1, respectively, in the **termios** *c_oflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

vt0 vt1 Select the style of delay for vertical-tabs. This shall have the effect of setting VTDLY to VT0 or VT1, respectively, in the **termios** *c_oflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

Local Modes

isig (**–isig**) Enable (disable) the checking of characters against the special control characters INTR, QUIT, and SUSP. This shall have the effect of setting (not setting) ISIG in the **termios** *c_lflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

icanon (**–icanon**) Enable (disable) canonical input (ERASE and KILL processing). This shall have the effect of setting (not setting) ICANON in the **termios** *c_lflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

iexten (**–iexten**) Enable (disable) any implementation-defined special control characters not currently controlled by **icanon**, **isig**, **ixon**, or **ixoff**. This shall have the effect of setting (not setting) IEXTEN in the **termios** *c_lflag* field, as defined in the Base

Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

echo (**–echo**) Echo back (do not echo back) every character typed. This shall have the effect of setting (not setting) ECHO in the **termios** *c_lflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

echoe (**–echoe**) The ERASE character visually erases (does not erase) the last character in the current line from the display, if possible. This shall have the effect of setting (not setting) ECHOE in the **termios** *c_lflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

echok (**–echok**) Echo (do not echo) NL after KILL character. This shall have the effect of setting (not setting) ECHOK in the **termios** *c_lflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

echonl (**–echonl**) Echo (do not echo) NL, even if **echo** is disabled. This shall have the effect of setting (not setting) ECHONL in the **termios** *c_lflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

noflsh (**–noflsh**) Disable (enable) flush after INTR, QUIT, SUSP. This shall have the effect of setting (not setting) NOFLSH in the **termios** *c_lflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

tostop (**–tostop**) Send SIGTTOU for background output. This shall have the effect of setting (not setting) TOSTOP in the **termios** *c_lflag* field, as defined in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface.

Special Control Character Assignments

<control>-character string

Set *<control>-character* to *string*. If *<control>-character* is one of the character sequences in the first column of the following table, the corresponding Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface control character from the second column shall be recognized. This has the effect of setting the corresponding element of the **termios** *c_cc* array (see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 13, Headers, **<termios.h>**).

34216 **Table 4-19** Control Character Names in *stty*

34217 | Control Character | c_cc Subscript | Description |
|---|---|---|
34218 | **eof** | VEOF | EOF character |
34219 | **eol** | VEOL | EOL character |
34220 | **erase** | VERASE | ERASE character |
34221 | **intr** | VINTR | INTR character |
34222 | **kill** | VKILL | KILL character |
34223 | **quit** | VQUIT | QUIT character |
34224 | **susp** | VSUSP | SUSP character |
34225 | **start** | VSTART | START character |
34226 | **stop** | VSTOP | STOP character |

34227 If *string* is a single character, the control character shall be set to that character. If *string* is
34228 the two-character sequence "^–" or the string *undef*, the control character shall be set to
34229 _POSIX_VDISABLE , if it is in effect for the device; if _POSIX_VDISABLE is not in effect for
34230 the device, it shall be treated as an error. In the POSIX locale, if *string* is a two-character
34231 sequence beginning with circumflex (`'^'`), and the second character is one of those listed in
34232 the "^c" column of the following table, the control character shall be set to the
34233 corresponding character value in the Value column of the table.

34234 **Table 4-20** Circumflex Control Characters in *stty*

34235 | ^c | Value | ^c | Value | ^c | Value |
|---|---|---|---|---|---|
34236 | a, A | <SOH> | l, L | <FF> | w, W | <ETB> |
34237 | b, B | <STX> | m, M | <CR> | x, X | <CAN> |
34238 | c, C | <ETX> | n, N | <SO> | y, Y | <EM> |
34239 | d, D | <EOT> | o, O | <SI> | z, Z | <SUB> |
34240 | e, E | <ENQ> | p, P | <DLE> | [| <ESC> |
34241 | f, F | <ACK> | q, Q | <DC1> | \ | <FS> |
34242 | g, G | <BEL> | r, R | <DC2> |] | <GS> |
34243 | h, H | <BS> | s, S | <DC3> | ^ | <RS> |
34244 | i, I | <HT> | t, T | <DC4> | _ | <US> |
34245 | j, J | <LF> | u, U | <NAK> | ? | <DEL> |
34246 | k, K | <VT> | v, V | <SYN> | | |

34247 **min** *number*
34248 Set the value of MIN to *number*. MIN is used in non-canonical mode input processing
34249 (**icanon**).

34250 **time** *number*
34251 Set the value of TIME to *number*. TIME is used in non-canonical mode input processing
34252 (**icanon**).

34253 **Combination Modes**

34254 *saved settings*
34255 Set the current terminal characteristics to the saved settings produced by the –**g** option.

34256 **evenp** or **parity**
34257 Enable **parenb** and **cs7**; disable **parodd**.

34258 **oddp**
34259 Enable **parenb**, **cs7**, and **parodd**.

34260 –**parity**, –**evenp**, or –**oddp**
34261 Disable **parenb**, and set **cs8**.

34262 XSI **raw** (–**raw** or **cooked**)
34263 Enable (disable) raw input and output. Raw mode shall be equivalent to setting:

```
34264     stty cs8 erase ^– kill ^– intr ^– \
34265         quit ^– eof ^– eol ^– –post –inpck
```

34266 **nl** (–**nl**)
34267 Enable (disable) **icrnl**. In addition, –**nl** unsets **inlcr** and **igncr**.

34268 **ek** Reset ERASE and KILL characters back to system defaults.

34269 **sane** Reset all modes to some reasonable, unspecified, values.

34270 **STDIN**

34271 Although no input is read from standard input, standard input shall be used to get the current
34272 terminal I/O characteristics and to set new terminal I/O characteristics.

34273 **INPUT FILES**

34274 None.

34275 **ENVIRONMENT VARIABLES**

34276 The following environment variables shall affect the execution of *stty*:

34277 *LANG* Provide a default value for the internationalization variables that are unset or null.
34278 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
34279 Internationalization Variables for the precedence of internationalization variables
34280 used to determine the values of locale categories.)

34281 *LC_ALL* If set to a non-empty string value, override the values of all the other
34282 internationalization variables.

34283 *LC_CTYPE* This variable determines the locale for the interpretation of sequences of bytes of
34284 text data as characters (for example, single-byte as opposed to multi-byte
34285 characters in arguments) and which characters are in the class **print**.

34286 *LC_MESSAGES*
34287 Determine the locale that should be used to affect the format and contents of
34288 diagnostic messages written to standard error.

34289 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

34290 **ASYNCHRONOUS EVENTS**

34291 Default.

34292 **STDOUT**

34293 If operands are specified, no output shall be produced.

34294 If the –**g** option is specified, *stty* shall write to standard output the current settings in a form that
34295 can be used as arguments to another instance of *stty* on the same system.

34296 If the –**a** option is specified, all of the information as described in the OPERANDS section shall
34297 be written to standard output. Unless otherwise specified, this information shall be written as
34298 <space>-separated tokens in an unspecified format, on one or more lines, with an unspecified
34299 number of tokens per line. Additional information may be written.

34300 If no options or operands are specified, an unspecified subset of the information written for the
34301 –**a** option shall be written.

If speed information is written as part of the default output, or if the –**a** option is specified and if the terminal input speed and output speed are the same, the speed information shall be written as follows:

```
"speed %d baud;", <speed>
```

Otherwise, speeds shall be written as:

```
"ispeed %d baud; ospeed %d baud;", <ispeed>, <ospeed>
```

In locales other than the POSIX locale, the word **baud** may be changed to something more appropriate in those locales.

If control characters are written as part of the default output, or if the –**a** option is specified, control characters shall be written as:

```
"%s = %s;", <control-character name>, <value>
```

where <*value*> is either the character, or some visual representation of the character if it is non-printable, or the string *undef* if the character is disabled.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 The terminal options were read or set successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The –**g** flag is designed to facilitate the saving and restoring of terminal state from the shell level. For example, a program may:

```
saveterm="$(stty –g)"       # save terminal state
stty (new settings)         # set new state
...                        # ...
stty $saveterm              # restore terminal state
```

Since the format is unspecified, the saved value is not portable across systems.

Since the –**a** format is so loosely specified, scripts that save and restore terminal settings should use the –**g** option.

EXAMPLES

None.

RATIONALE

The original *stty* description was taken directly from System V and reflected the System V terminal driver **termio**. It has been modified to correspond to the terminal driver **termios**.

Output modes are specified only for XSI-conformant systems. All implementations are expected to provide *stty* operands corresponding to all of the output modes they support.

The *stty* utility is primarily used to tailor the user interface of the terminal, such as selecting the preferred ERASE and KILL characters. As an application programming utility, *stty* can be used within shell scripts to alter the terminal settings for the duration of the script.

The **termios** section states that individual disabling of control characters is possible through the option _POSIX_VDISABLE. If enabled, two conventions currently exist for specifying this: System V uses "^−", and BSD uses *undef*. Both are accepted by *stty* in this volume of IEEE Std 1003.1-200x. The other BSD convention of using the letter 'u' was rejected because it conflicts with the actual letter 'u', which is an acceptable value for a control character.

Early proposals did not specify the mapping of "^c" to control characters because the control characters were not specified in the POSIX locale character set description file requirements. The control character set is now specified in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 3, Definitions so the historical mapping is specified. Note that although the mapping corresponds to control-character key assignments on many terminals that use the ISO/IEC 646:1991 standard (or ASCII) character encodings, the mapping specified here is to the control characters, not their keyboard encodings.

Since **termios** supports separate speeds for input and output, two new options were added to specify each distinctly.

Some historical implementations use standard input to get and set terminal characteristics; others use standard output. Since input from a login TTY is usually restricted to the owner while output to a TTY is frequently open to anyone, using standard input provides fewer chances of accidentally (or maliciously) altering the terminal settings of other users. Using standard input also allows *stty* **−a** and *stty* **−g** output to be redirected for later use. Therefore, usage of standard input is required by this volume of IEEE Std 1003.1-200x.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface

CHANGE HISTORY

First released in Issue 2.

Issue 5

The description of **tabs** is clarified.

FUTURE DIRECTIONS section added.

Issue 6

The legacy items **iuclc**(**−iuclc**), **xcase**, **olcuc**(**−olcuc**), **lcase**(**−lcase**), and **LCASE**(**−LCASE**), are removed.

NAME

tabs — set terminal tabs

SYNOPSIS

UP XSI `tabs [ –n| –a| –a2| –c| –c2| –c3| –f| –p| –s| –u][+m[n]] [–T type]`

`tabs [–T type][ +[n]] n1[,n2,...]`

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 and optionally adjusts the margin.

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.

OPTIONS

The *tabs* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines, except for various extensions: the options **–a2**, **–c2**, and **–c3** are multi-character.

The following options shall be supported:

–*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.

XSI **–a** 1,10,16,36,72
Assembler, applicable to some mainframes.

XSI **–a2** 1,10,16,40,72
Assembler, applicable to some mainframes.

XSI **–c** 1,8,12,16,20,55
COBOL, normal format.

XSI **–c2** 1,6,10,14,49
COBOL, compact format (columns 1 to 6 omitted).

XSI **–c3** 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1 to 6 omitted), with more tabs than **–c2**.

XSI **–f** 1,7,11,15,19,23
FORTRAN

XSI **–p** 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
PL/1

XSI **–s** 1,10,55
SNOBOL

XSI **–u** 1,12,20,44
Assembler, applicable to some mainframes.

34423 –**T** *type* Indicate the type of terminal. If this option is not supplied and the *TERM* variable
34424 is unset or null, an unspecified default terminal type shall be used. The setting of
34425 *type* shall take precedence over the value in *TERM*.

34426 **OPERANDS**

34427 The following operand shall be supported:

34428 *n1***[**,*n2*,...**]** A single command line argument that consists of tab-stop values separated using
34429 either commas or <blank>s. The application shall ensure that the tab-stop values
34430 are positive decimal integers in strictly ascending order. If any number (except the
34431 first one) is preceded by a plus sign, it is taken as an increment to be added to the
34432 previous value. For example, the tab lists 1,10,20,30 and 1,10,+10,+10 are
34433 considered to be identical.

34434 **STDIN**

34435 Not used.

34436 **INPUT FILES**

34437 None.

34438 **ENVIRONMENT VARIABLES**

34439 The following environment variables shall affect the execution of *tabs*:

34440 *LANG* Provide a default value for the internationalization variables that are unset or null.
34441 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
34442 Internationalization Variables for the precedence of internationalization variables
34443 used to determine the values of locale categories.)

34444 *LC_ALL* If set to a non-empty string value, override the values of all the other
34445 internationalization variables.

34446 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
34447 characters (for example, single-byte as opposed to multi-byte characters in
34448 arguments).

34449 *LC_MESSAGES*
34450 Determine the locale that should be used to affect the format and contents of
34451 diagnostic messages written to standard error.

34452 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

34453 *TERM* Determine the terminal type. If this variable is unset or null, and if the –**T** option is
34454 not specified, an unspecified default terminal type shall be used.

34455 **ASYNCHRONOUS EVENTS**

34456 Default.

34457 **STDOUT**

34458 If standard output is a terminal, the appropriate sequence to clear and set the tab stops may be
34459 written to standard output in an unspecified format. If standard output is not a terminal,
34460 undefined results occur.

34461 **STDERR**

34462 The standard error shall be used only for diagnostic messages.

34463 **OUTPUT FILES**

34464 None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

This utility makes use of the terminal's hardware tabs and the *stty tabs* option.

This utility is not recommended for application use.

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* works if standard output is directed to the terminal; if output is directed to another file, however, *tabs* fails.

EXAMPLES

None.

RATIONALE

Consideration was given to having the *tput* utility handle all of the functions described in *tabs*. 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 Shell and Utilities *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 volume of IEEE Std 1003.1-200x has a reduced feature list, but many of the features omitted were restored as XSI extensions even though the supported languages and coding styles are primarily historical.

There was considerable sentiment for specifying only a means of resetting the tabs back to a known state—presumably the ''standard'' of tabs every eight positions. The following features were omitted:

- 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.

In an early proposal, 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 tab stops of the terminal is not a capability in historical *terminfo* or *termcap* facilities and might not be supported on a wide range of terminals.

FUTURE DIRECTIONS

None.

SEE ALSO

expand, *stty*, *unexpand*

34505 **CHANGE HISTORY**

34506 First released in Issue 2.

34507 **Issue 6**

34508 This utility is now marked as part of the User Portability Utilities option.

34509 The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

NAME

tail — copy the last part of a file

SYNOPSIS

```
tail [–f][ –c number| –n number][file]
```

DESCRIPTION

The *tail* utility shall copy its input file to the standard output beginning at a designated place.

Copying shall begin at the point in the file indicated by the **–c** *number* or **–n** *number* options. The option-argument *number* shall be counted in units of lines or bytes, according to the options **–n** and **–c**. Both line and byte counts start from 1.

Tails relative to the end of the file may be saved in an internal buffer, and thus may be limited in length. Such a buffer, if any, shall be no smaller than {LINE_MAX}*10 bytes.

OPTIONS

The *tail* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–c *number* The application shall ensure that the *number* option-argument is a decimal integer whose sign affects the location in the file, measured in bytes, to begin the copying:

Sign	Copying Starts
+	Relative to the beginning of the file.
–	Relative to the end of the file.
none	Relative to the end of the file.

The origin for counting shall be 1; that is, **–c** +1 represents the first byte of the file, **–c** –1 the last.

–f If the input file is a regular file or if the *file* operand specifies a FIFO, do not terminate after the last line of the input file has been copied, but read and copy further bytes from the input file when they become available. If no *file* operand is specified and standard input is a pipe, the **–f** option shall be ignored. If the input file is not a FIFO, pipe, or regular file, it is unspecified whether or not the **–f** option shall be ignored.

–n *number* This option shall be equivalent to **–c** *number*, except the starting location in the file shall be measured in lines instead of bytes. The origin for counting shall be 1; that is, **–n** +1 represents the first line of the file, **–n** –1 the last.

If neither **–c** nor **–n** is specified, **–n** 10 shall be assumed.

OPERANDS

The following operand shall be supported:

file A pathname of an input file. If no *file* operands are specified, the standard input shall be used.

STDIN

The standard input shall be used only if no *file* operands are specified. See the INPUT FILES section.

34550 **INPUT FILES**

34551 If the **–c** option is specified, the input file can contain arbitrary data; otherwise, the input file
34552 shall be a text file.

34553 **ENVIRONMENT VARIABLES**

34554 The following environment variables shall affect the execution of *tail*:

34555 *LANG* Provide a default value for the internationalization variables that are unset or null.
34556 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
34557 Internationalization Variables for the precedence of internationalization variables
34558 used to determine the values of locale categories.)

34559 *LC_ALL* If set to a non-empty string value, override the values of all the other
34560 internationalization variables.

34561 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
34562 characters (for example, single-byte as opposed to multi-byte characters in
34563 arguments and input files).

34564 *LC_MESSAGES*
34565 Determine the locale that should be used to affect the format and contents of
34566 diagnostic messages written to standard error.

34567 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

34568 **ASYNCHRONOUS EVENTS**

34569 Default.

34570 **STDOUT**

34571 The designated portion of the input file shall be written to standard output.

34572 **STDERR**

34573 The standard error shall be used only for diagnostic messages.

34574 **OUTPUT FILES**

34575 None.

34576 **EXTENDED DESCRIPTION**

34577 None.

34578 **EXIT STATUS**

34579 The following exit values shall be returned:

34580 0 Successful completion.

34581 >0 An error occurred.

34582 **CONSEQUENCES OF ERRORS**

34583 Default.

34584 **APPLICATION USAGE**

34585 The **–c** option should be used with caution when the input is a text file containing multi-byte
34586 characters; it may produce output that does not start on a character boundary.

34587 Although the input file to *tail* can be any type, the results might not be what would be expected
34588 on some character special device files or on file types not described by the System Interfaces
34589 volume of IEEE Std 1003.1-200x. Since this volume of IEEE Std 1003.1-200x does not specify the
34590 block size used when doing input, *tail* need not read all of the data from devices that only
34591 perform block transfers.

EXAMPLES

The **–f** option can be used to monitor the growth of a file that is being written by some other process. For example, the command:

```
tail –f fred
```

prints the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed. As another example, the command:

```
tail –f –c 15 fred
```

prints the last 15 bytes of the file **fred**, followed by any bytes that are appended to **fred** between the time *tail* is initiated and killed.

RATIONALE

This version of *tail* was created to allow conformance to the Utility Syntax Guidelines. The historical **–b** option was omitted because of the general non-portability of block-sized units of text. The **–c** option historically meant ‘‘characters’’, but this volume of IEEE Std 1003.1-200x indicates that it means ‘‘bytes’’. This was selected to allow reasonable implementations when multi-byte characters are possible; it was not named **–b** to avoid confusion with the historical **–b**.

The origin of counting both lines and bytes is 1, matching all widespread historical implementations.

The restriction on the internal buffer is a compromise between the historical System V implementation of 4 096 bytes and the BSD 32 768 bytes.

The **–f** option has been implemented as a loop that sleeps for 1 second and copies any bytes that are available. This is sufficient, but if more efficient methods of determining when new data are available are developed, implementations are encouraged to use them.

Historical documentation indicates that *tail* ignores the **–f** option if the input file is a pipe (pipe and FIFO on systems that support FIFOs). On BSD-based systems, this has been true; on System V-based systems, this was true when input was taken from standard input, but it did not ignore the **–f** flag if a FIFO was named as the *file* operand. Since the **–f** option is not useful on pipes and all historical implementations ignore **–f** if no *file* operand is specified and standard input is a pipe, this volume of IEEE Std 1003.1-200x requires this behavior. However, since the **–f** option is useful on a FIFO, this volume of IEEE Std 1003.1-200x also requires that if standard input is a FIFO or a FIFO is named, the **–f** option shall not be ignored. Although historical behavior does not ignore the **–f** option for other file types, this is unspecified so that implementations are allowed to ignore the **–f** option if it is known that the file cannot be extended.

This was changed to the current form based on comments noting that **–c** was almost never used without specifying a number and that there was no need to specify **–l** if **–n** *number* was given.

FUTURE DIRECTIONS

None.

SEE ALSO

head

CHANGE HISTORY

First released in Issue 2.

Issue 6

The obsolescent SYNOPSIS lines and associated text are removed.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

NAME

talk — talk to another user

SYNOPSIS

UP

```
talk address [terminal]
```

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 processed as follows:

- 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 **termios** interface in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal 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 cause those characters to be sent to the recipient's terminal.
- When and only when the *stty* **iexten** local mode is enabled, the existence and processing of additional special control characters and multi-byte or single-byte functions shall be implementation-defined.
- Typing other non-printable 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. However, a user's privilege may further constrain the domain of accessibility of other users' terminals. The *talk* utility shall fail when the user lacks the appropriate privileges to perform the requested action.

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.

OPTIONS

None.

34676 **OPERANDS**

34677 The following operands shall be supported:

34678 *address* The recipient of the *talk* session. One form of *address* is the <*user name*>, as returned
34679 by the *who* utility. Other address formats and how they are handled are
34680 unspecified.

34681 *terminal* If the recipient is logged in more than once, the *terminal* argument can be used to
34682 indicate the appropriate terminal name. If *terminal* is not specified, the *talk* message
34683 shall be displayed on one or more accessible terminals in use by the recipient. The
34684 format of *terminal* shall be the same as that returned by the *who* utility.

34685 **STDIN**

34686 Characters read from standard input shall be copied to the recipient's terminal in an unspecified
34687 manner. If standard input is not a terminal, talk shall write a diagnostic message and exit with a
34688 non-zero status.

34689 **INPUT FILES**

34690 None.

34691 **ENVIRONMENT VARIABLES**

34692 The following environment variables shall affect the execution of *talk*:

34693 *LANG* Provide a default value for the internationalization variables that are unset or null.
34694 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
34695 Internationalization Variables for the precedence of internationalization variables
34696 used to determine the values of locale categories.)

34697 *LC_ALL* If set to a non-empty string value, override the values of all the other
34698 internationalization variables.

34699 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
34700 characters (for example, single-byte as opposed to multi-byte characters in
34701 arguments and input files). If the recipient's locale does not use an *LC_CTYPE*
34702 equivalent to the sender's, the results are undefined.

34703 *LC_MESSAGES*
34704 Determine the locale that should be used to affect the format and contents of
34705 diagnostic messages written to standard error and informative messages written to
34706 standard output.

34707 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

34708 *TERM* Determine the name of the invoker's terminal type. If this variable is unset or null,
34709 an unspecified default terminal type shall be used.

34710 **ASYNCHRONOUS EVENTS**

34711 When the *talk* utility receives a SIGINT signal, the utility shall terminate and exit with a zero
34712 status. It shall take the standard action for all other signals.

34713 **STDOUT**

34714 If standard output is a terminal, characters copied from the recipient's standard input may be
34715 written to standard output. Standard output also may be used for diagnostic messages. If
34716 standard output is not a terminal, *talk* shall exit with a non-zero status.

34717 **STDERR**

34718 None.

34719 **OUTPUT FILES**

34720 None.

34721 **EXTENDED DESCRIPTION**

34722 None.

34723 **EXIT STATUS**

34724 The following exit values shall be returned:

34725 0 Successful completion.

34726 >0 An error occurred or *talk* was invoked on a terminal incapable of supporting it.

34727 **CONSEQUENCES OF ERRORS**

34728 Default.

34729 **APPLICATION USAGE**

34730 Because the handling of non-printable, non-<space>s is tied to the *stty* description of **iexten**,
34731 implementation extensions within the terminal driver can be accessed. For example, some
34732 implementations provide line editing functions with certain control character sequences.

34733 **EXAMPLES**

34734 None.

34735 **RATIONALE**

34736 The *write* utility was included in this volume of IEEE Std 1003.1-200x since it can be
34737 implemented on all terminal types. The *talk* utility, which cannot be implemented on certain
34738 terminals, was considered to be a ''better'' communications interface. Both of these programs are
34739 in widespread use on historical implementations. Therefore, both utilities have been specified.

34740 All references to networking abilities (*talk*ing to a user on another system) were removed as
34741 being outside the scope of this volume of IEEE Std 1003.1-200x.

34742 Historical BSD and System V versions of *talk* terminate both of the conversations when either
34743 user breaks out of the session. This can lead to adverse consequences if a user unwittingly
34744 continues to enter text that is interpreted by the shell when the other terminates the session.
34745 Therefore, the version of *talk* specified by this volume of IEEE Std 1003.1-200x requires both
34746 users to terminate their end of the session explicitly.

34747 Only messages sent to the terminal of the invoking user can be internationalized in any way:

34748 • The original ''Message from <*unspecified string*> ...'' message sent to the terminal of the
34749 recipient cannot be internationalized because the environment of the recipient is as yet
34750 inaccessible to the *talk* utility. The environment of the invoking party is irrelevant.

34751 • Subsequent communication between the two parties cannot be internationalized because the
34752 two parties may specify different languages in their environment (and non-portable
34753 characters cannot be mapped from one language to another).

34754 • Neither party can be required to communicate in a language other than C and/or the one
34755 specified by their environment because unavailable terminal hardware support (for example,
34756 fonts) may be required.

34757 The text in the STDOUT section reflects the usage of the verb ''display'' in this section; some *talk*
34758 implementations actually use standard output to write to the terminal, but this volume of
34759 IEEE Std 1003.1-200x does not require that to be the case.

34760 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*
34761 require that they all use or accept the same format.

The handling of non-printable 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 non-printable characters that may send commands to the other terminal.

FUTURE DIRECTIONS

None.

SEE ALSO

mesg, *who*, *write*, the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

34775 **NAME**

34776 tee — duplicate standard input

34777 **SYNOPSIS**

34778 `tee [–ai][`*`file`*`...]`

34779 **DESCRIPTION**

34780 The *tee* utility shall copy standard input to standard output, making a copy in zero or more files.
34781 The *tee* utility shall not buffer output.

34782 If the **–a** option is not specified, output files shall be written (see Section 1.7.1.4 (on page 2204).

34783 **OPTIONS**

34784 The *tee* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
34785 Utility Syntax Guidelines.

34786 The following options shall be supported:

34787 **–a** Append the output to the files.

34788 **–i** Ignore the SIGINT signal.

34789 **OPERANDS**

34790 The following operands shall be supported:

34791 *file* A pathname of an output file. Processing of at least 13 *file* operands shall be
34792 supported.

34793 **STDIN**

34794 The standard input can be of any type.

34795 **INPUT FILES**

34796 None.

34797 **ENVIRONMENT VARIABLES**

34798 The following environment variables shall affect the execution of *tee*:

34799 *LANG* Provide a default value for the internationalization variables that are unset or null.
34800 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
34801 Internationalization Variables for the precedence of internationalization variables
34802 used to determine the values of locale categories.)

34803 *LC_ALL* If set to a non-empty string value, override the values of all the other
34804 internationalization variables.

34805 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
34806 characters (for example, single-byte as opposed to multi-byte characters in
34807 arguments).

34808 *LC_MESSAGES*
34809 Determine the locale that should be used to affect the format and contents of
34810 diagnostic messages written to standard error.

34811 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

34812 **ASYNCHRONOUS EVENTS**

34813 Default, except that if the **–i** option was specified, SIGINT shall be ignored.

34814 **STDOUT**

34815 The standard output shall be a copy of the standard input.

34816 **STDERR**

34817 The standard error shall be used only for diagnostic messages.

34818 **OUTPUT FILES**

34819 If any *file* operands are specified, the standard input shall be copied to each named file.

34820 **EXTENDED DESCRIPTION**

34821 None.

34822 **EXIT STATUS**

34823 The following exit values shall be returned:

34824 0 The standard input was successfully copied to all output files.

34825 >0 An error occurred.

34826 **CONSEQUENCES OF ERRORS**

34827 If a write to any successfully opened *file* operand fails, writes to other successfully opened *file*
34828 operands and standard output shall continue, but the exit status shall be non-zero. Otherwise,
34829 the default actions specified in Section 1.11 (on page 2221) apply.

34830 **APPLICATION USAGE**

34831 The *tee* utility is usually used in a pipeline, to make a copy of the output of some utility.

34832 The *file* operand is technically optional, but *tee* is no more useful than *cat* when none is specified.

34833 **EXAMPLES**

34834 Save an unsorted intermediate form of the data in a pipeline:

34835 ```
... | tee unsorted | sort > sorted
```

34836 **RATIONALE**

34837 The buffering requirement means that *tee* is not allowed to use ISO C standard fully buffered or
34838 line-buffered writes. It does not mean that *tee* has to do 1-byte reads followed by 1-byte writes.

34839 It should be noted that early versions of BSD ignore any invalid options and accept a single '–'
34840 as an alternative to –**i**. They also print a message if unable to open a file:

34841 `"tee: cannot access %s\n", <pathname>`

34842 Historical implementations ignore write errors. This is explicitly not permitted by this volume of
34843 IEEE Std 1003.1-200x.

34844 Some historical implementations use O_APPEND when providing append mode; others use the
34845 *lseek*( ) function to seek to the end of file after opening the file without O_APPEND. This volume
34846 of IEEE Std 1003.1-200x requires functionality equivalent to using O_APPEND; see Section
34847 1.7.1.4 (on page 2204).

34848 **FUTURE DIRECTIONS**

34849 None.

34850 **SEE ALSO**

34851 *cat*

34852 **CHANGE HISTORY**

34853 First released in Issue 2.
```

Issue 6

IEEE PASC Interpretation 1003.2 #168 is applied.

NAME

test — evaluate expression

SYNOPSIS

```
test [expression]
[ [expression] ]
```

DESCRIPTION

The *test* utility shall evaluate the *expression* and indicates the result of the evaluation by its exit status. An exit status of zero indicates that the expression evaluated as true and an exit status of 1 indicates that the expression evaluated as false.

In the second form of the utility, which uses "[]" rather than *test*, the application shall ensure that the square brackets are separate arguments.

OPTIONS

The *test* utility shall not recognize the "−−" argument in the manner specified by guideline 10 in the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

No options shall be supported.

OPERANDS

The application shall ensure that all operators and elements of primaries are presented as separate arguments to the *test* utility.

The following primaries can be used to construct *expression*:

−b *file* True if *file* exists and is a block special file.

−c *file* True if *file* exists and is a character special file.

−d *file* True if *file* exists and is a directory.

−e *file* True if *file* exists.

−f *file* True if *file* exists and is a regular file.

−g *file* True if *file* exists and its set-group-ID flag is set.

−h *file* True if *file* exists and is a symbolic link.

−L *file* True if *file* exists and is a symbolic link.

−n *string* True if the length of *string* is non-zero.

−p *file* True if *file* is a FIFO.

−r *file* True if *file* exists and is readable. True shall indicate that permission to read from *file* will be granted, as defined in Section 1.7.1.4 (on page 2204).

−S *file* True if *file* exists and is a socket.

−s *file* True if *file* exists and has a size greater than zero.

−t *file_descriptor*
True if the file whose file descriptor number is *file_descriptor* is open and is associated with a terminal.

−u *file* True if *file* exists and its set-user-ID flag is set.

−w *file* True if *file* exists and is writable. True shall indicate that permission to write from *file* will be granted, as defined in Section 1.7.1.4 (on page 2204).

34895 **–x** *file* True if *file* exists and is executable. True shall indicate that permission to execute |
34896 *file* will be granted, as defined in Section 1.7.1.4 (on page 2204). If *file* is a directory, |
34897 true shall indicate that permission to search *file* will be granted.

34898 **–z** *string* True if the length of string *string* is zero.

34899 *string* True if the string *string* is not the null string.

34900 *s1* **=** *s2* True if the strings *s1* and *s2* are identical.

34901 *s1* **!=** *s2* True if the strings *s1* and *s2* are not identical.

34902 *n1* **–eq** *n2* True if the integers *n1* and *n2* are algebraically equal.

34903 *n1* **–ne** *n2* True if the integers *n1* and *n2* are not algebraically equal.

34904 *n1* **–gt** *n2* True if the integer *n1* is algebraically greater than the integer *n2*.

34905 *n1* **–ge** *n2* True if the integer *n1* is algebraically greater than or equal to the integer *n2*.

34906 *n1* **–lt** *n2* True if the integer *n1* is algebraically less than the integer *n2*.

34907 *n1* **–le** *n2* True if the integer *n1* is algebraically less than or equal to the integer *n2*.

34908 XSI *expression1* **–a** *expression2*
34909 True if both *expression1* and *expression2* are true. The **–a** binary primary is left
34910 associative. It has a higher precedence than **–o**.

34911 XSI *expression1* **–o** *expression2*
34912 True if either *expression1* or *expression2* is true. The **–o** binary primary is left
34913 associative.

34914 With the exception of the **–h** *file* primary, if a *file* argument is a symbolic link, *test* shall evaluate
34915 the expression by resolving the symbolic link and using the file referenced by the link.

34916 These primaries can be combined with the following operators:

34917 **!** *expression* True if *expression* is false.

34918 XSI **(** *expression* **)** True if *expression* is true. The parentheses can be used to alter the normal
34919 precedence and associativity.

34920 The primaries with two elements of the form:

34921 *–primary_operator primary_operand*

34922 are known as *unary primaries*. The primaries with three elements in either of the two forms:

34923 *primary_operand –primary_operator primary_operand*

34924 *primary_operand primary_operator primary_operand*

34925 are known as *binary primaries*. Additional implementation-defined operators and
34926 *primary_operator*s may be provided by implementations. They shall be of the form *–operator*
34927 where the first character of *operator* is not a digit.

34928 The algorithm for determining the precedence of the operators and the return value that shall be
34929 generated is based on the number of arguments presented to *test*. (However, when using the
34930 "`[...]`" form, the right-bracket final argument shall not be counted in this algorithm.)

34931 In the following list, $1, $2, $3, and $4 represent the arguments presented to *test*:

34932 0 arguments: Exit false (1).

	1 argument:	Exit true (0) if $1 is not null; otherwise, exit false.
	2 arguments:	• If $1 is `'!'`, exit true if $2 is null, false if $2 is not null.
		• If $1 is a unary primary, exit true if the unary test is true, false if the unary test is false.
		• Otherwise, produce unspecified results.
	3 arguments:	• If $2 is a binary primary, perform the binary test of $1 and $3.
		• If $1 is `'!'`, negate the two-argument test of $2 and $3.
		• If $1 is `'('` and $3 is `')'`, perform the unary test of $2.
		• Otherwise, produce unspecified results.
	4 arguments:	• If $1 is `'!'`, negate the three-argument test of $2, $3, and $4.
XSI		• If $1 is `'('` and $4 is `')'`, perform the two-argument test of $2 and $3.
		• Otherwise, the results are unspecified.
	>4 arguments:	The results are unspecified.
XSI		On XSI-conformant systems, combinations of primaries and operators shall be evaluated using the precedence and associativity rules described previously. In addition, the string comparison binary primaries `'='` and `"!="` shall have a higher precedence than any unary primary.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *test*:

LANG	Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)
LC_ALL	If set to a non-empty string value, override the values of all the other internationalization variables.
LC_CTYPE	Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).
LC_MESSAGES	Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.
XSI *NLSPATH*	Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

34971 **STDOUT**
34972 Not used.

34973 **STDERR**
34974 The standard error shall be used only for diagnostic messages.

34975 **OUTPUT FILES**
34976 None.

34977 **EXTENDED DESCRIPTION**
34978 None.

34979 **EXIT STATUS**
34980 The following exit values shall be returned:

34981 0 *expression* evaluated to true.

34982 1 *expression* evaluated to false or *expression* was missing.

34983 >1 An error occurred.

34984 **CONSEQUENCES OF ERRORS**
34985 Default.

34986 **APPLICATION USAGE**
34987 Scripts should be careful when dealing with user-supplied input that could be confused with
34988 primaries and operators. Unless the application writer knows all the cases that produce input to
34989 the script, invocations like:

34990 `test "$1" –a "$2"`

34991 should be written as:

34992 `test "$1" && test "$2"`

34993 to avoid problems if a user supplied values such as $1 set to `'!'` and $2 set to the null string.
34994 That is, in cases where maximal portability is of concern, replace:

34995 `test expr1 –a expr2`

34996 with:

34997 `test expr1 && test expr2`

34998 and replace:

34999 `test expr1 –o expr2`

35000 with:

35001 `test expr1 || test expr2`

35002 but note that, in *test*, **–a** has higher precedence than **–o** while `"&&"` and `"||"` have equal
35003 precedence in the shell.

35004 Parentheses or braces can be used in the shell command language to effect grouping.

35005 Parentheses must be escaped when using *sh*; for example:

35006 `test \( expr1 –a expr2 \) –o expr3`

35007 This command is not always portable outside XSI-conformant systems. The following form can
35008 be used instead:

```
( test expr1 && test expr2 ) || test expr3
```

The two commands:

```
test "$1"
test ! "$1"
```

could not be used reliably on some historical systems. Unexpected results would occur if such a *string* expression were used and $1 expanded to `'!'`, `'('`, or a known unary primary. Better constructs are:

```
test –n "$1"
test –z "$1"
```

respectively.

Historical systems have also been unreliable given the common construct:

```
test "$response" = "expected string"
```

One of the following is a more reliable form:

```
test "X$response" = "Xexpected string"
test "expected string" = "$response"
```

Note that the second form assumes that *expected string* could not be confused with any unary primary. If *expected string* starts with `'–'`, `'('`, `'!'`, or even `'='`, the first form should be used instead. Using the preceding rules without the XSI marked extensions, any of the three comparison forms is reliable, given any input. (However, note that the strings are quoted in all cases.)

Because the string comparison binary primaries, `'='` and `"!="`, have a higher precedence than any unary primary in the greater than 4 argument case, unexpected results can occur if arguments are not properly prepared. For example, in:

```
test –d $1 –o –d $2
```

If $1 evaluates to a possible directory name of `'='`, the first three arguments are considered a string comparison, which shall cause a syntax error when the second **–d** is encountered. One of the following forms prevents this; the second is preferred:

```
test \( –d "$1" \) –o \( –d "$2" \)
test –d "$1" || test –d "$2"
```

Also in the greater than 4 argument case:

```
test "$1" = "bat" –a "$2" = "ball"
```

Syntax errors occur if $1 evaluates to `'('` or `'!'`. One of the following forms prevents this; the third is preferred:

```
test "X$1" = "Xbat" –a "X$2" = "Xball"
test "$1" = "bat" && test "$2" = "ball"
test "X$1" = "Xbat" && test "X$2" = "Xball"
```

EXAMPLES

1. Exit if there are not two or three arguments (two variations):

```
if [ $# –ne 2 –a $# –ne 3 ]; then exit 1; fi
if [ $# –lt 2 –o $# –gt 3 ]; then exit 1; fi
```

2. Perform a *mkdir* if a directory does not exist:

```
test ! –d tempdir && mkdir tempdir
```

3. Wait for a file to become non-readable:

```
while test –r thefile
do
    sleep 30
done
echo '"thefile" is no longer readable'
```

4. Perform a command if the argument is one of three strings (two variations):

```
if [ "$1" = "pear" ] || [ "$1" = "grape" ] || [ "$1" = "apple" ]
then
    command
fi
```

```
case "$1" in
    pear|grape|apple) command ;;
esac
```

RATIONALE

The KornShell-derived conditional command (double bracket **[[ ]]**) was removed from the shell command language description in an early proposal. Objections were raised that the real problem is misuse of the *test* command (**[**), and putting it into the shell is the wrong way to fix the problem. Instead, proper documentation and a new shell reserved word (**!**) are sufficient.

Tests that require multiple *test* operations can be done at the shell level using individual invocations of the *test* command and shell logicals, rather than using the error-prone **–o** flag of *test*.

XSI-conformant systems support more than four arguments.

XSI-conformant systems support the combining of primaries with the following constructs:

expression1 **–a** *expression2*
: True if both *expression1* and *expression2* are true.

expression1 **–o** *expression2*
: True if at least one of *expression1* and *expression2* are true.

(*expression* **)**
: True if *expression* is true.

In evaluating these more complex combined expressions, the following precedence rules are used:

- The unary primaries have higher precedence than the algebraic binary primaries.
- The unary primaries have lower precedence than the string binary primaries.
- The unary and binary primaries have higher precedence than the unary *string* primary.
- The **!** operator has higher precedence than the **–a** operator, and the **–a** operator has higher precedence than the **–o** operator.
- The **–a** and **–o** operators are left associative.
- The parentheses can be used to alter the normal precedence and associativity.

The BSD and System V versions of **–f** are not the same. The BSD definition was:

–f *file* True if *file* exists and is not a directory.

The SVID version (true if the file exists and is a regular file) was chosen for this volume of IEEE Std 1003.1-200x because its use is consistent with the **–b**, **–c**, **–d**, and **–p** operands (*file* exists and is a specific file type).

The **–e** primary, possessing similar functionality to that provided by the C shell, was added because it provides the only way for a shell script to find out if a file exists without trying to open the file. Since implementations are allowed to add additional file types, a portable script cannot use:

```
test –b foo –o –c foo –o –d foo –o –f foo –o –p foo
```

to find out if **foo** is an existing file.) On historical BSD systems, the existence of a file could be determined by:

```
test –f foo –o –d foo
```

but there was no easy way to determine that an existing file was a regular file. An early proposal used the KornShell **–a** primary (with the same meaning), but this was changed to **–e** because there were concerns about the high probability of humans confusing the **–a** primary with the **–a** binary operator.

The following options were not included in this volume of IEEE Std 1003.1-200x, although they are provided by some implementations. These operands should not be used by new implementations for other purposes:

–k *file* True if *file* exists and its sticky bit is set.

–C *file* True if *file* is a contiguous file.

–V *file* True if *file* is a version file.

The following option was not included because it was undocumented in most implementations, has been removed from some implementations (including System V), and the functionality is provided by the shell (see Section 2.6.2 (on page 2239).

–l *string* The length of the string *string*.

The **–b**, **–c**, **–g**, **–p**, **–u**, and **–x** operands are derived from the SVID; historical BSD does not provide them. The **–k** operand is derived from System V; historical BSD does not provide it.

On historical BSD systems, *test* **–w** *directory* always returned false because *test* tried to open the directory for writing, which always fails.

Some additional primaries newly invented or from the KornShell appeared in an early proposal as part of the conditional command (**[[ ]]**): *s1* **>** *s2*, *s1* **<** *s2*, *str* **=** *pattern*, *str* **!=** *pattern*, *f1* **–nt** *f2*, *f1* **–ot** *f2*, and *f1* **–ef** *f2*. They were not carried forward into the *test* utility when the conditional command was removed from the shell because they have not been included in the *test* utility built into historical implementations of the *sh* utility.

The **–t** *file_descriptor* primary is shown with a mandatory argument because the grammar is ambiguous if it can be omitted. Historical implementations have allowed it to be omitted, providing a default of 1.

35129 **FUTURE DIRECTIONS**

35130 None.

35131 **SEE ALSO**

35132 *find*

35133 **CHANGE HISTORY**

35134 First released in Issue 2.

35135 **Issue 5**

35136 FUTURE DIRECTIONS section added.

35137 **Issue 6**

35138 The **–h** operand is added for symbolic links, and access permission requirements are clarified for
35139 the **–r**, **–w**, and **–x** operands to align with the IEEE P1003.2b draft standard.

35140 The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

35141 The **–L** and **–S** operands are added for symbolic links and sockets.

NAME

time — time a simple command

SYNOPSIS

UP

```
time [–p] utility [argument...]
```

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 *times*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x 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 *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 (for example, 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, except when it is the sole command within a grouping command (see Section 2.9.4.1 (on page 2253)) in that pipeline. For example, the commands on the left are unspecified; those on the right report on utilities **a** and **c**, respectively:

```
time a | b | c     { time a } | b | c
a | b | time c     a | b | (time c)
```

OPTIONS

The *time* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following option shall be supported:

–p Write the timing output to standard error in the format shown in the STDERR section.

OPERANDS

The following operands shall be supported:

utility The name of a utility that is to be invoked. If the *utility* operand names any of the special built-in utilities in Section 2.14 (on page 2266), the results are undefined.

argument Any string to be supplied as an argument when invoking the utility named by the *utility* operand.

STDIN

Not used.

INPUT FILES

None.

35182 **ENVIRONMENT VARIABLES**

35183 The following environment variables shall affect the execution of *time*:

35184 *LANG* Provide a default value for the internationalization variables that are unset or null.
35185 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
35186 Internationalization Variables for the precedence of internationalization variables
35187 used to determine the values of locale categories.)

35188 *LC_ALL* If set to a non-empty string value, override the values of all the other
35189 internationalization variables.

35190 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
35191 characters (for example, single-byte as opposed to multi-byte characters in
35192 arguments).

35193 *LC_MESSAGES*
35194 Determine the locale that should be used to affect the format and contents of
35195 diagnostic and informative messages written to standard error.

35196 *LC_NUMERIC*
35197 Determine the locale for numeric formatting.

35198 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

35199 *PATH* Determine the search path that shall be used to locate the utility to be invoked; see
35200 the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment
35201 Variables.

35202 **ASYNCHRONOUS EVENTS**

35203 Default.

35204 **STDOUT**

35205 Not used.

35206 **STDERR**

35207 The standard error shall be used to write the timing statistics. If **−p** is specified, the following
35208 format shall be used in the POSIX locale:

35209 `"real %f\nuser %f\nsys %f\n", <real seconds>, <user seconds>,`
35210 `<system seconds>`

35211 where each floating-point number shall be expressed in seconds. The precision used may be less
35212 than the default six digits of `%f`, but shall be sufficiently precise to accommodate the size of the
35213 clock tick on the system (for example, if there were 60 clock ticks per second, at least two digits
35214 shall follow the radix character). The number of digits following the radix character shall be no
35215 less than one, even if this always results in a trailing zero. The implementation may append
35216 white space and additional information following the format shown here.

35217 **OUTPUT FILES**

35218 None.

35219 **EXTENDED DESCRIPTION**

35220 None.

35221 **EXIT STATUS**

35222 If the *utility* utility is invoked, the exit status of *time* shall be the exit status of *utility*; otherwise,
35223 the *time* utility shall exit with one of the following values:

35224 1-125 An error occurred in the *time* utility.

126 The utility specified by *utility* was found but could not be invoked.

127 The utility specified by *utility* could not be found.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if an error occurs so that applications can distinguish ''failure to find a utility'' from ''invoked utility exited with an error indication''. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for ''normal error conditions'' and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for any other reason.

EXAMPLES

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'
```

RATIONALE

When the *time* utility was originally proposed to be included in the earlier version of IEEE Std 1003.1, questions were raised about its suitability for inclusion on the grounds that it was not useful for portable applications, specifically:

- The underlying CPU definitions from the System Interfaces volume of IEEE Std 1003.1-200x are 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 scope this volume of IEEE Std 1003.1-200x.

However, *time* does fit in the scope of user portability. Human judgement 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. Hence it has been included in this volume of IEEE Std 1003.1-200x.

The default output format has been left unspecified because historical implementations differ greatly in their style of depicting this numeric output. The **-p** option was invented to provide scripts a common means of obtaining this information.

In the KornShell, *time* is a shell reserved word that can be used to time an entire pipeline, rather than just a simple command. The POSIX definition has been 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* utility 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 historical KornShell usage.

The term *utility* is used, rather than *command*, to highlight the fact that shell compound commands, pipelines, special built-ins, and so on, cannot be used directly. However, *utility*

includes user application programs and shell scripts, not just the standard utilities.

FUTURE DIRECTIONS

None.

SEE ALSO

sh, the System Interfaces volume of IEEE Std 1003.1-200x, *times*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

NAME

touch — change file access and modification times

SYNOPSIS

```
touch [–acm][ –r ref_file| –t time] file...
```

DESCRIPTION

The *touch* utility shall change the modification times, access times, or both of files. The modification time shall be equivalent to the value of the *st_mtime* member of the **stat** structure for a file, as described in the System Interfaces volume of IEEE Std 1003.1-200x; the access time shall be equivalent to the value of *st_atime*.

The time used can be specified by the **–t** *time* option-argument, the corresponding time fields of the file referenced by the **–r** *ref_file* option-argument, or the *date_time* operand, as specified in the following sections. If none of these are specified, *touch* shall use the current time (the value returned by the equivalent of the *time*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x).

For each *file* operand, *touch* shall perform actions equivalent to the following functions defined in the System Interfaces volume of IEEE Std 1003.1-200x:

1. If *file* does not exist, a *creat*( ) function call is made with the *file* operand used as the *path* argument and the value of the bitwise-inclusive OR of S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH used as the *mode* argument.
2. The *utime*( ) function is called with the following arguments:
   a. The *file* operand is used as the *path* argument.
   b. The **utimbuf** structure members *actime* and *modtime* are determined as described in the OPTIONS section.

OPTIONS

The *touch* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–a Change the access time of *file*. Do not change the modification time unless **–m** is also specified.

–c Do not create a specified *file* if it does not exist. Do not write any diagnostic messages concerning this condition.

–m Change the modification time of *file*. Do not change the access time unless **–a** is also specified.

–r *ref_file* Use the corresponding time of the file named by the pathname *ref_file* instead of the current time.

–t *time* Use the specified *time* instead of the current time. The option-argument shall be a decimal number of the form:

```
[[CC]YY]MMDDhhmm[.SS]
```

where each two digits represents the following:

MM The month of the year [01,12].

DD The day of the month [01,31].

hh The hour of the day [00,23].

mm The minute of the hour [00,59].

CC The first two digits of the year (the century).

YY The second two digits of the year.

SS The second of the minute [00,60].

Both *CC* and *YY* shall be optional. If neither is given, the current year shall be assumed. If *YY* is specified, but *CC* is not, *CC* shall be derived as follows:

If *YY* is:	***CC* becomes:**
[69,99]	19
[00,68]	20

Note: It is expected that in a future version of IEEE Std 1003.1-200x the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)

The resulting time shall be affected by the value of the *TZ* environment variable. If the resulting time value precedes the Epoch, *touch* shall exit immediately with an error status. The range of valid times past the Epoch is implementation-defined, but it shall extend to at least the time 0 hours, 0 minutes, 0 seconds, January 1, 2038, Coordinated Universal Time. Some implementations may not be able to represent dates beyond the January 18, 2038, because they use **signed int** as a time holder.

The range for *SS* is [00,60] rather than [00,59] because of leap seconds. If *SS* is 60, and the resulting time, as affected by the *TZ* environment variable, does not refer to a leap second, the resulting time shall be one second after a time where *SS* is 59. If *SS* is not given a value, it is assumed to be zero.

If neither the **–a** nor **–m** options were specified, *touch* shall behave as if both the **–a** and **–m** options were specified.

OPERANDS

The following operands shall be supported:

file A pathname of a file whose times shall be modified.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *touch*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in

arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

TZ Determine the timezone to be used for interpreting the *time* option-argument. If *TZ* is unset or null, an unspecified default timezone shall be used.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 The utility executed successfully and all requested changes were made.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The interpretation of time is taken to be *seconds since the Epoch* (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 4.14, Seconds Since the Epoch). It should be noted that implementations conforming to the System Interfaces volume of IEEE Std 1003.1-200x do not take leap seconds into account when computing seconds since the Epoch. When *SS*=60 is used, the resulting time always refers to 1 plus *seconds since the Epoch* for a time when *SS*=59.

Although the **–t** *time* option-argument specifies values in 1969, the access time and modification time fields are defined in terms of seconds since the Epoch (00:00:00 on 1 January 1970 UTC). Therefore, depending on the value of *TZ* when *touch* is run, there is never more than a few valid hours in 1969 and there need not be any valid times in 1969.

One ambiguous situation occurs if **–t** *time* is not specified, **–r** *ref_file* is not specified, and the first operand is an eight or ten-digit decimal number. A portable script can avoid this problem by using:

```
touch -- file
```

or:

```
touch ./file
```

in this case.

EXAMPLES

None.

RATIONALE

The functionality of *touch* is described almost entirely through references to functions in the System Interfaces volume of IEEE Std 1003.1-200x. In this way, there is no duplication of effort required for describing such side effects as the relationship of user IDs to the user database, permissions, and so on.

There are some significant differences between the *touch* utility in this volume of IEEE Std 1003.1-200x and those in System V and BSD systems. They are upward-compatible for historical applications from both implementations:

1. In System V, an ambiguity exists when a pathname that is a decimal number leads the operands; it is treated as a time value. In BSD, no *time* value is allowed; files may only be *touch*ed to the current time. The **–t** *time* construct solves these problems for future conforming applications (note that the **–t** option is not historical practice).
2. The inclusion of the century digits, *CC*, is also new. Note that a ten-digit *time* value is treated as if *YY*, and not *CC*, were specified. The caveat about the range of dates following the Epoch was included as recognition that some implementations are not able to represent dates beyond 18 January 2038 because they use **signed int** as a time holder.

The **–r** option was added because several comments requested this capability. This option was named **–f** in an early proposal, but was changed because the **–f** option is used in the BSD version of *touch* with a different meaning.

At least one historical implementation of *touch* incremented the exit code if **–c** was specified and the file did not exist. This volume of IEEE Std 1003.1-200x requires exit status zero if no errors occur.

FUTURE DIRECTIONS

Applications should use the **–r** or **–t** options.

SEE ALSO

date, the System Interfaces volume of IEEE Std 1003.1-200x, *creat*( ), *time*( ), **<sys/stat.h>**

CHANGE HISTORY

First released in Issue 2.

Issue 6

The obsolescent *date_time* operand is removed.

The Open Group Corrigendum U027/1 is applied. This extends the range of valid time past the Epoch to at least the time 0 hours, 0 minutes, 0 seconds, January 1, 2038, Coordinated Universal Time. This is a new requirement on POSIX implementations.

The range for double leap seconds is changed from [00,61] to [00,60] to align with the ISO/IEC 9899: 1999 standard.

NAME

tput — change terminal characteristics

SYNOPSIS

UP

```
tput [–T type] operand...
```

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.

OPTIONS

The *tput* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following option shall be supported:

–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*.

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 initializes the user's terminal in an implementation-defined manner.

reset Display the sequence that resets 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.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *tput*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

35485 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

35486 *TERM* Determine the terminal type. If this variable is unset or null, and if the **–T** option is
35487 not specified, an unspecified default terminal type shall be used.

35488 **ASYNCHRONOUS EVENTS**
35489 Default.

35490 **STDOUT**
35491 If standard output is a terminal device, it may be used for writing the appropriate sequence to
35492 clear the screen or reset or initialize the terminal. If standard output is not a terminal device,
35493 undefined results occur.

35494 **STDERR**
35495 The standard error shall be used only for diagnostic messages.

35496 **OUTPUT FILES**
35497 None.

35498 **EXTENDED DESCRIPTION**
35499 None.

35500 **EXIT STATUS**
35501 The following exit values shall be returned:

35502 0 The requested string was written successfully.

35503 1 Unspecified.

35504 2 Usage error.

35505 3 No information is available about the specified terminal type.

35506 4 The specified operand is invalid.

35507 >4 An error occurred.

35508 **CONSEQUENCES OF ERRORS**
35509 If one of the operands is not available for the terminal, *tput* continues processing the remaining
35510 operands.

35511 **APPLICATION USAGE**
35512 The difference between resetting and initializing a terminal is left unspecified, as they vary
35513 greatly based on hardware types. In general, resetting is a more severe action.

35514 Some terminals use control characters to perform the stated functions, and on such terminals it
35515 might make sense to use *tput* to store the initialization strings in a file or environment variable
35516 for later use. However, because other terminals might rely on system calls to do this work, the
35517 standard output cannot be used in a portable manner, such as the following non-portable
35518 constructs:

35519 ```
ClearVar=‘tput clear‘
```
35520 ```
tput reset | mailx –s "Wake Up" ddg
```

35521 **EXAMPLES**

35522 1. Initialize the terminal according to the type of terminal in the environmental variable
35523 *TERM*. This command can be included in a **.profile** file.

35524 ```
tput init
```

35525 2. Reset a 450 terminal.
```

```
tput –T 450 reset
```

RATIONALE

The list of operands was reduced to a minimum for the following reasons:

- The only features chosen were those that were likely to be used by human users interacting with a terminal.
- Specifying the full *terminfo* set was not considered desirable, but the standard developers did not want to select among operands.
- This volume of IEEE Std 1003.1-200x does not attempt to provide applications with sophisticated 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 this varies greatly based on hardware types. In general, resetting is a more severe action.

The exit status of 1 is historically reserved for finding out if a Boolean operand is not set. Although the operands were reduced to a minimum, the exit status of 1 should still be reserved for the Boolean operands, for those sites that wish to support them.

FUTURE DIRECTIONS

None.

SEE ALSO

stty, *tabs*

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

35549 **NAME**

35550 tr — translate characters

35551 **SYNOPSIS**

35552 `tr [-c | -C][-s] string1 string2`

35553 `tr -s [-c | -C] string1`

35554 `tr -d [-c | -C] string1`

35555 `tr -ds [-c | -C] string1 string2`

35556 **DESCRIPTION**

35557 The *tr* utility shall copy the standard input to the standard output with substitution or deletion
35558 of selected characters. The options specified and the *string1* and *string2* operands shall control
35559 translations that occur while copying characters and single-character collating elements.

35560 **OPTIONS**

35561 The *tr* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
35562 Utility Syntax Guidelines.

35563 The following options shall be supported:

35564 **-c** Complement the set of values specified by *string1*. See the EXTENDED
35565 DESCRIPTION section.

35566 **-C** Complement the set of characters specified by *string1*. See the EXTENDED
35567 DESCRIPTION section.

35568 **-d** Delete all occurrences of input characters that are specified by *string1*.

35569 **-s** Replace instances of repeated characters with a single character, as described in the
35570 EXTENDED DESCRIPTION section.

35571 **OPERANDS**

35572 The following operands shall be supported:

35573 *string1*, *string2*
35574 Translation control strings. Each string shall represent a set of characters to be
35575 converted into an array of characters used for the translation. For a detailed
35576 description of how the strings are interpreted, see the EXTENDED DESCRIPTION
35577 section.

35578 **STDIN**

35579 The standard input can be any type of file.

35580 **INPUT FILES**

35581 None.

35582 **ENVIRONMENT VARIABLES**

35583 The following environment variables shall affect the execution of *tr*:

35584 *LANG* Provide a default value for the internationalization variables that are unset or null.
35585 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
35586 Internationalization Variables for the precedence of internationalization variables
35587 used to determine the values of locale categories.)

35588 *LC_ALL* If set to a non-empty string value, override the values of all the other
35589 internationalization variables.

35590 *LC_COLLATE*
35591 Determine the locale for the behavior of range expressions and equivalence classes.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments) and the behavior of character classes.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The *tr* output shall be identical to the input, with the exception of the specified transformations.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

The operands *string1* and *string2* (if specified) define two arrays of characters. The constructs in the following list can be used to specify characters or single-character collating elements. If any of the constructs result in multi-character collating elements, *tr* shall exclude, without a diagnostic, those multi-character elements from the resulting array.

character Any character not described by one of the conventions below shall represent itself.

\octal Octal sequences can be used to represent characters with specific coded values. An octal sequence shall consist of a backslash followed by the longest sequence of one, two, or three-octal-digit characters (01234567). The sequence shall cause the value whose encoding is represented by the one, two, or three-digit octal integer to be placed into the array. If the size of a byte on the system is greater than nine bits, the valid escape sequence used to represent a byte is implementation-defined. Multi-byte characters require multiple, concatenated escape sequences of this type, including the leading `'\'` for each byte.

\character The backslash-escape sequences in the Base Definitions volume of IEEE Std 1003.1-200x, Table 5-1, Escape Sequences and Associated Actions (`'\\'`, `'\a'`, `'\b'`, `'\f'`, `'\n'`, `'\r'`, `'\t'`, `'\v'`) shall be supported. The results of using any other character, other than an octal digit, following the backslash are unspecified.

c–c In the POSIX locale, this construct shall represent the range of collating elements between the range endpoints (as long as neither endpoint is an octal sequence of the form *\octal*), inclusive, as defined by the collation sequence. The characters or collating elements in the range shall be placed in the array in ascending collation sequence. If the second endpoint precedes the starting endpoint in the collation sequence, it is unspecified whether the range of collating elements is empty, or this construct is treated as invalid. In locales other than the POSIX locale, this construct has unspecified behavior.

If either or both of the range endpoints are octal sequences of the form *\octal*, this shall represent the range of specific coded values between the two range endpoints, inclusive.

[:*class*:] Represents all characters belonging to the defined character class, as defined by the current setting of the *LC_CTYPE* locale category. The following character class names shall be accepted when specified in *string1*:

alnum	**blank**	**digit**	**lower**	**punct**	**upper**
alpha	**cntrl**	**graph**	**print**	**space**	**xdigit**

XSI In addition, character class expressions of the form [:*name*:] shall be recognized in those locales where the *name* keyword has been given a **charclass** definition in the *LC_CTYPE* category.

When both the **–d** and **–s** options are specified, any of the character class names shall be accepted in *string2*. Otherwise, only character class names **lower** or **upper** are valid in *string2* and then only if the corresponding character class (**upper** and **lower**, respectively) is specified in the same relative position in *string1*. Such a specification shall be interpreted as a request for case conversion. When [: *lower*:] appears in *string1* and [:*upper*:] appears in *string2*, the arrays shall contain the characters from the **toupper** mapping in the *LC_CTYPE* category of the current locale. When [:*upper*:] appears in *string1* and [:*lower*:] appears in *string2*, the arrays shall contain the characters from the **tolower** mapping in the *LC_CTYPE* category of the current locale. The first character from each mapping pair shall be in the array for *string1* and the second character from each mapping pair shall be in the array for *string2* in the same relative position.

Except for case conversion, the characters specified by a character class expression shall be placed in the array in an unspecified order.

If the name specified for *class* does not define a valid character class in the current locale, the behavior is undefined.

[=*equiv*=] Represents all characters or collating elements belonging to the same equivalence class as *equiv*, as defined by the current setting of the *LC_COLLATE* locale category. An equivalence class expression shall be allowed only in *string1*, or in *string2* when it is being used by the combined **–d** and **–s** options. The characters belonging to the equivalence class shall be placed in the array in an unspecified order.

[*x***n*] Represents *n* repeated occurrences of the character *x*. Because this expression is used to map multiple characters to one, it is only valid when it occurs in *string2*. If *n* is omitted or is zero, it shall be interpreted as large enough to extend the *string2*-based sequence to the length of the *string1*-based sequence. If *n* has a leading zero, it shall be interpreted as an octal value. Otherwise, it shall be interpreted as a decimal value.

When the **–d** option is not specified:

- Each input character found in the array specified by *string1* shall be replaced by the character in the same relative position in the array specified by *string2*. When the array specified by *string2* is shorter that the one specified by *string1*, the results are unspecified.
- If the **–C** option is specified, the complements of the characters specified by *string1* (the set of all characters in the current character set, as defined by the current setting of *LC_CTYPE*, except for those actually specified in the *string1* operand) shall be placed in the array in ascending collation sequence, as defined by the current setting of *LC_COLLATE*.
- If the **–c** option is specified, the complement of the values specified by *string1* shall be placed in the array in ascending order by binary value.

- Because the order in which characters specified by character class expressions or equivalence class expressions is undefined, such expressions should only be used if the intent is to map several characters into one. An exception is case conversion, as described previously.

When the **–d** option is specified:

- Input characters found in the array specified by *string1* shall be deleted.
- When the **–C** option is specified with **–d**, all characters except those specified by *string1* shall be deleted. The contents of *string2* are ignored, unless the **–s** option is also specified.
- When the **–c** option is specified with **–d**, all values except those specified by *string1* shall be deleted. The contents of *string2* shall be ignored, unless the **–s** option is also specified.
- The same string cannot be used for both the **–d** and the **–s** option; when both options are specified, both *string1* (used for deletion) and *string2* (used for squeezing) shall be required.

When the **–s** option is specified, after any deletions or translations have taken place, repeated sequences of the same character shall be replaced by one occurrence of the same character, if the character is found in the array specified by the last operand. If the last operand contains a character class, such as the following example:

```
tr –s '[:space:]'
```

the last operand's array shall contain all of the characters in that character class. However, in a case conversion, as described previously, such as:

```
tr –s '[:upper:]' '[:lower:]'
```

the last operand's array shall contain only those characters defined as the second characters in each of the **toupper** or **tolower** character pairs, as appropriate.

An empty string used for *string1* or *string2* produces undefined results.

EXIT STATUS

The following exit values shall be returned:

0 All input was processed successfully.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

If necessary, *string1* and *string2* can be quoted to avoid pattern matching by the shell.

If an ordinary digit (representing itself) is to follow an octal sequence, the octal sequence must use the full three digits to avoid ambiguity.

When *string2* is shorter than *string1*, a difference results between historical System V and BSD systems. A BSD system pads *string2* with the last character found in *string2*. Thus, it is possible to do the following:

```
tr 0123456789 d
```

which would translate all digits to the letter `'d'`. Since this area is specifically unspecified in this volume of IEEE Std 1003.1-200x, both the BSD and System V behaviors are allowed, but a conforming application cannot rely on the BSD behavior. It would have to code the example in the following way:

```
tr 0123456789 '[d*]'
```

It should be noted that, despite similarities in appearance, the string operands used by *tr* are not regular expressions.

Unlike some historical implementations, this definition of the *tr* utility correctly processes NUL characters in its input stream. NUL characters can be stripped by using:

```
tr –d '\000'
```

EXAMPLES

1. The following example creates a list of all words in **file1** one per line in **file2**, where a word is taken to be a maximal string of letters.

   ```
   tr –cs "[:alpha:]" "[\n*]" <file1 >file2
   ```

2. The next example translates all lowercase characters in **file1** to uppercase and writes the results to standard output.

   ```
   tr "[:lower:]" "[:upper:]" <file1
   ```

3. This example uses an equivalence class to identify accented variants of the base character `'e'` in **file1**, which are stripped of diacritical marks and written to **file2**.

   ```
   tr "[=e=]" e <file1 >file2
   ```

RATIONALE

In some early proposals, an explicit option **–n** was added to disable the historical behavior of stripping NUL characters from the input. It was considered that automatically stripping NUL characters from the input was not correct functionality. However, the removal of **–n** in a later proposal does not remove the requirement that *tr* correctly process NUL characters in its input stream. NUL characters can be stripped by using *tr* **–d** '\000'.

Historical implementations of *tr* differ widely in syntax and behavior. For example, the BSD version has not needed the bracket characters for the repetition sequence. The POSIX Shell and Utilities *tr* syntax is based more closely on the System V and XPG3 model while attempting to accommodate historical BSD implementations. In the case of the short *string2* padding, the decision was to unspecify the behavior and preserve System V and XPG3 scripts, which might find difficulty with the BSD method. The assumption was made that BSD users of *tr* have to make accommodations to meet the POSIX Shell and Utilities syntax. Since it is possible to use the repetition sequence to duplicate the desired behavior, whereas there is no simple way to achieve the System V method, this was the correct, if not desirable, approach.

The use of octal values to specify control characters, while having historical precedents, is not portable. The introduction of escape sequences for control characters should provide the necessary portability. It is recognized that this may cause some historical scripts to break.

An early proposal included support for multi-character collating elements. It was pointed out that, while *tr* does employ some syntactical elements from REs, the aim of *tr* is quite different; ranges, for example, do not have a similar meaning (''any of the chars in the range matches'', *versus* ''translate each character in the range to the output counterpart''). As a result, the previously included support for multi-character collating elements has been removed. What remains are ranges in current collation order (to support, for example, accented characters), character classes, and equivalence classes.

In XPG3 the [:*class*:] and [=*equiv*=] conventions are shown with double brackets, as in RE syntax. However, *tr* does not implement RE principles; it just borrows part of the syntax. Consequently, [:*class*:] and [=*equiv*=] should be regarded as syntactical elements on a par with [*x***n*], which is not an RE bracket expression.

The standard developers will consider changes to *tr* that allow it to translate characters between different character encodings, or they will consider providing a new utility to accomplish this.

On historical System V systems, a range expression requires enclosing square-brackets, such as:

```
tr '[a-z]' '[A-Z]'
```

However, BSD-based systems did not require the brackets, and this convention is used by POSIX Shell and Utilities to avoid breaking large numbers of BSD scripts:

```
tr a-z A-Z
```

The preceding System V script will continue to work because the brackets, treated as regular characters, are translated to themselves. However, any System V script that relied on *a-z* representing the three characters '–,' and 'z' have to be rewritten as *az*–.

A prior version of IEEE Std 1003.1-200x had a **–c** option that behaved similarly to the **–C** option, but did not supply functionality equivalent to the **–c** option specified in IEEE Std 1003.1-200x. This meant that historical practice of being able to specify *tr* **–d** \200–\377 (which would delete all bytes with the top bit set) would have no effect because, in the C locale, bytes with the values octal 200 to octal 377 are not characters.

The earlier version also said that octal sequences referred to collating elements and could be placed adjacent to each other to specify multi-byte characters. However, it was noted that this caused ambiguities because *tr* would not be able to tell whether adjacent octal sequences were intending to specify multi-byte characters or multiple single byte characters. IEEE Std 1003.1-200x specifies that octal sequences always refer to single byte binary values.

FUTURE DIRECTIONS

None.

SEE ALSO

sed

CHANGE HISTORY

First released in Issue 2.

Issue 6

The **–C** operand is added, and the description of the **–c** operand is changed to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

35798 **NAME**

35799 true — return true value

35800 **SYNOPSIS**

35801 `true`

35802 **DESCRIPTION**

35803 The *true* utility shall return with exit code zero.

35804 **OPTIONS**

35805 None.

35806 **OPERANDS**

35807 None.

35808 **STDIN**

35809 Not used.

35810 **INPUT FILES**

35811 None.

35812 **ENVIRONMENT VARIABLES**

35813 None.

35814 **ASYNCHRONOUS EVENTS**

35815 Default.

35816 **STDOUT**

35817 Not used.

35818 **STDERR**

35819 None.

35820 **OUTPUT FILES**

35821 None.

35822 **EXTENDED DESCRIPTION**

35823 None.

35824 **EXIT STATUS**

35825 Default.

35826 **CONSEQUENCES OF ERRORS**

35827 None.

35828 **APPLICATION USAGE**

35829 This utility is typically used in shell scripts, as shown in the EXAMPLES section. The special
35830 built-in utility `:` is sometimes more efficient than *true*.

35831 **EXAMPLES**

35832 This command is executed forever:

```
35833 while true
35834 do
35835     command
35836 done
```

35837 **RATIONALE**

35838 The *true* utility has been retained in this volume of IEEE Std 1003.1-200x, even though the shell
35839 special built-in : provides similar functionality, because *true* is widely used in historical scripts
35840 and is less cryptic to novice script readers.

35841 **FUTURE DIRECTIONS**

35842 None.

35843 **SEE ALSO**

35844 *false*, Section 2.9 (on page 2248)

35845 **CHANGE HISTORY**

35846 First released in Issue 2.

35847 **NAME**

35848 tsort — topological sort

35849 **SYNOPSIS**

35850 XSI
```
tsort [file]
```
35851

35852 **DESCRIPTION**

35853 The *tsort* utility shall write to standard output a totally ordered list of items consistent with a
35854 partial ordering of items contained in the input.

35855 The application shall ensure that the input consists of pairs of items (non-empty strings)
35856 separated by <blank>s. Pairs of different items indicate ordering. Pairs of identical items
35857 indicate presence, but not ordering.

35858 **OPTIONS**

35859 None.

35860 **OPERANDS**

35861 The following operand shall be supported:

35862 *file* A pathname of a text file to order. If no *file* operand is given, the standard input |
35863 shall be used. |

35864 **STDIN**

35865 The standard input shall be a text file that is used if no *file* operand is given.

35866 **INPUT FILES**

35867 The input file named by the *file* operand is a text file.

35868 **ENVIRONMENT VARIABLES**

35869 The following environment variables shall affect the execution of *tsort*:

35870 *LANG* Provide a default value for the internationalization variables that are unset or null.
35871 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
35872 Internationalization Variables for the precedence of internationalization variables
35873 used to determine the values of locale categories.)

35874 *LC_ALL* If set to a non-empty string value, override the values of all the other
35875 internationalization variables.

35876 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
35877 characters (for example, single-byte as opposed to multi-byte characters in
35878 arguments and input files).

35879 *LC_MESSAGES*
35880 Determine the locale that should be used to affect the format and contents of
35881 diagnostic messages written to standard error.

35882 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

35883 **ASYNCHRONOUS EVENTS**

35884 Default.

35885 **STDOUT**

35886 The standard output shall be a text file consisting of the order list produced from the partially
35887 ordered input.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The *LC_COLLATE* variable need not affect the actions of *tsort*. The output ordering is not lexicographic, but depends on the pairs of items given as input.

EXAMPLES

The command:

```
tsort <<EOF
a b c c d e
g g
f g e f
h h
EOF
```

produces the output:

a
b
c
d
e
f
g
h

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

None.

CHANGE HISTORY

First released in Issue 2.

Issue 6

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

35930 **NAME**

35931 tty — return user's terminal name

35932 **SYNOPSIS**

35933 `tty`

35934 **DESCRIPTION**

35935 The *tty* utility shall write to the standard output the name of the terminal that is open as
35936 standard input. The name that is used shall be equivalent to the string that would be returned by
35937 the *ttyname*( ) function defined in the System Interfaces volume of IEEE Std 1003.1-200x.

35938 **OPTIONS**

35939 The *tty* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
35940 Utility Syntax Guidelines.

35941 **OPERANDS**

35942 None.

35943 **STDIN**

35944 While no input is read from standard input, standard input shall be examined to determine
35945 whether or not it is a terminal, and, if so, to determine the name of the terminal.

35946 **INPUT FILES**

35947 None.

35948 **ENVIRONMENT VARIABLES**

35949 The following environment variables shall affect the execution of *tty*:

35950 *LANG* Provide a default value for the internationalization variables that are unset or null.
35951 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
35952 Internationalization Variables for the precedence of internationalization variables
35953 used to determine the values of locale categories.)

35954 *LC_ALL* If set to a non-empty string value, override the values of all the other
35955 internationalization variables.

35956 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
35957 characters (for example, single-byte as opposed to multi-byte characters in
35958 arguments).

35959 *LC_MESSAGES*
35960 Determine the locale that should be used to affect the format and contents of
35961 diagnostic messages written to standard error and informative messages written to
35962 standard output.

35963 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

35964 **ASYNCHRONOUS EVENTS**

35965 Default.

35966 **STDOUT**

35967 If standard input is a terminal device, a pathname of the terminal as specified by the *ttyname*( )
35968 function defined in the System Interfaces volume of IEEE Std 1003.1-200x shall be written in the
35969 following format:

35970 `"%s\n", <terminal name>`

35971 Otherwise, a message shall be written indicating that standard input is not connected to a
35972 terminal. In the POSIX locale, the *tty* utility shall use the format:

```
"not a tty\n"
```

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Standard input is a terminal.

1 Standard input is not a terminal.

>1 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

This utility checks the status of the file open as standard input against that of a implementation-defined set of files. It is possible that no match can be found, or that the match found need not be the same file as that which was opened for standard input (although they are the same device).

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *isatty*( ), *ttyname*( )

CHANGE HISTORY

First released in Issue 2.

Issue 5

The SYNOPSIS is changed to indicate two forms of the command, with the second form marked as obsolete. This is a clarification and does not change the functionality published in previous issues.

Issue 6

The obsolescent **–s** option is removed.

36007 **NAME**

36008 type — write a description of command type

36009 **SYNOPSIS**

36010 XSI
```
type name...
```
36011

36012 **DESCRIPTION**

36013 The *type* utility shall indicate how each argument would be interpreted if used as a command
36014 name.

36015 **OPTIONS**

36016 None.

36017 **OPERANDS**

36018 The following operand shall be supported:

36019 *name* A name to be interpreted.

36020 **STDIN**

36021 Not used.

36022 **INPUT FILES**

36023 None.

36024 **ENVIRONMENT VARIABLES**

36025 The following environment variables shall affect the execution of *type*:

36026 *LANG* Provide a default value for the internationalization variables that are unset or null.
36027 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
36028 Internationalization Variables for the precedence of internationalization variables
36029 used to determine the values of locale categories.)

36030 *LC_ALL* If set to a non-empty string value, override the values of all the other
36031 internationalization variables.

36032 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
36033 characters (for example, single-byte as opposed to multi-byte characters in
36034 arguments).

36035 *LC_MESSAGES*
36036 Determine the locale that should be used to affect the format and contents of
36037 diagnostic messages written to standard error.

36038 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

36039 *PATH* Determine the location of *name*, as described in the Base Definitions volume of
36040 IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

36041 **ASYNCHRONOUS EVENTS**

36042 Default.

36043 **STDOUT**

36044 The standard output of *type* contains information about each operand in an unspecified format.
36045 The information provided typically identifies the operand as a shell built-in, function, alias, or
36046 keyword, and where applicable, may display the operand's pathname.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Since *type* must be aware of the contents of the current shell execution environment (such as the lists of commands, functions, and built-ins processed by *hash*), it is always provided as a shell regular built-in. If it is called in a separate utility execution environment, such as one of the following:

```
nohup type writer
find . –type f | xargs type
```

it might not produce accurate results.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

command

CHANGE HISTORY

First released in Issue 2.

36077 **NAME**

36078 ulimit — set or report file size limit

36079 **SYNOPSIS**

36080 XSI
```
ulimit [–f][blocks]
```
36081

36082 **DESCRIPTION**

36083 The *ulimit* utility shall set or report the file-size writing limit imposed on files written by the
36084 shell and its child processes (files of any size may be read). Only a process with appropriate
36085 privileges can increase the limit.

36086 **OPTIONS**

36087 The *ulimit* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
36088 12.2, Utility Syntax Guidelines.

36089 The following option shall be supported:

36090 **–f** Set (or report, if no *blocks* operand is present), the file size limit in blocks. The **–f**
36091 option shall also be the default case.

36092 **OPERANDS**

36093 The following operand shall be supported:

36094 *blocks* The number of 512-byte blocks to use as the new file size limit.

36095 **STDIN**

36096 Not used.

36097 **INPUT FILES**

36098 None.

36099 **ENVIRONMENT VARIABLES**

36100 The following environment variables shall affect the execution of *ulimit*:

36101 *LANG* Provide a default value for the internationalization variables that are unset or null.
36102 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
36103 Internationalization Variables for the precedence of internationalization variables
36104 used to determine the values of locale categories.)

36105 *LC_ALL* If set to a non-empty string value, override the values of all the other
36106 internationalization variables.

36107 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
36108 characters (for example, single-byte as opposed to multi-byte characters in
36109 arguments).

36110 *LC_MESSAGES*
36111 Determine the locale that should be used to affect the format and contents of
36112 diagnostic messages written to standard error.

36113 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

36114 **ASYNCHRONOUS EVENTS**

36115 Default.

36116 **STDOUT**

36117 The standard output shall be used when no *blocks* operand is present. If the current number of
36118 blocks is limited, the number of blocks in the current limit shall be written in the following
36119 format:

"%d\n", *<number of 512-byte blocks>*

If there is no current limit on the number of blocks, in the POSIX locale the following format shall be used:

"unlimited\n"

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 A request for a higher limit was rejected or an error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Since *ulimit* affects the current shell execution environment, it is always provided as a shell regular built-in. If it is called in separate utility execution environment, such as one of the following:

```
nohup ulimit –f 10000
env ulimit 10000
```

it does not affect the file size limit of the caller's environment.

Once a limit has been decreased by a process, it cannot be increased (unless appropriate privileges are involved), even back to the original system limit.

EXAMPLES

Set the file size limit to 51 200 bytes:

```
ulimit –f 100
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *ulimit*( )

CHANGE HISTORY

First released in Issue 2.

36156 **NAME**

36157 umask — get or set the file mode creation mask

36158 **SYNOPSIS**

36159 `umask [-S][`*`mask`*`]`

36160 **DESCRIPTION**

36161 The *umask* utility shall set the file mode creation mask of the current shell execution
36162 environment (see Section 2.12 (on page 2263)) to the value specified by the *mask* operand. This
36163 mask shall affect the initial value of the file permission bits of subsequently created files. If *umask*
36164 is called in a subshell or separate utility execution environment, such as one of the following:

36165 `(umask 002)`
36166 `nohup umask ...`
36167 `find . -exec umask ... \;`

36168 it shall not affect the file mode creation mask of the caller's environment.

36169 If the *mask* operand is not specified, the *umask* utility shall write to standard output the value of
36170 the invoking process's file mode creation mask.

36171 **OPTIONS**

36172 The *umask* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
36173 12.2, Utility Syntax Guidelines.

36174 The following option shall be supported:

36175 **-S** Produce symbolic output.

36176 The default output style is unspecified, but shall be recognized on a subsequent invocation of
36177 *umask* on the same system as a *mask* operand to restore the previous file mode creation mask.

36178 **OPERANDS**

36179 The following operand shall be supported:

36180 *mask* A string specifying the new file mode creation mask. The string is treated in the
36181 same way as the *mode* operand described in the EXTENDED DESCRIPTION |
36182 section for *chmod*. |

36183 For a *symbolic_mode* value, the new value of the file mode creation mask shall be
36184 the logical complement of the file permission bits portion of the file mode specified
36185 by the *symbolic_mode* string.

36186 In a *symbolic_mode* value, the permissions *op* characters `'+'` and `'-'` shall be
36187 interpreted relative to the current file mode creation mask; `'+'` shall cause the bits
36188 for the indicated permissions to be cleared in the mask; `'-'` shall cause the bits for
36189 the indicated permissions to be set in the mask.

36190 The interpretation of *mode* values that specify file mode bits other than the file
36191 permission bits is unspecified.

36192 In the octal integer form of *mode*, the specified bits are set in the file mode creation
36193 mask.

36194 The file mode creation mask shall be set to the resulting numeric value.

36195 The default output of a prior invocation of *umask* on the same system with no
36196 operand also shall be recognized as a *mask* operand.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *umask*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

When the *mask* operand is not specified, the *umask* utility shall write a message to standard output that can later be used as a *umask mask* operand.

If **–S** is specified, the message shall be in the following format:

```
"u=%s,g=%s,o=%s\n", <owner permissions>, <group permissions>,
    <other permissions>
```

where the three values shall be combinations of letters from the set {*r*, *w*, *x*}; the presence of a letter shall indicate that the corresponding bit is clear in the file mode creation mask.

If a *mask* operand is specified, there shall be no output written to standard output.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 The file mode creation mask was successfully changed, or no *mask* operand was supplied.

>0 An error occurred.

36237 **CONSEQUENCES OF ERRORS**

36238 Default.

36239 **APPLICATION USAGE**

36240 Since *umask* affects the current shell execution environment, it is generally provided as a shell
36241 regular built-in.

36242 In contrast to the negative permission logic provided by the file mode creation mask and the
36243 octal number form of the *mask* argument, the symbolic form of the *mask* argument specifies those
36244 permissions that are left alone.

36245 **EXAMPLES**

36246 Either of the commands:

36247 `umask a=rx,ug+w`

36248 `umask 002`

36249 sets the mode mask so that subsequently created files have their S_IWOTH bit cleared.

36250 After setting the mode mask with either of the above commands, the *umask* command can be
36251 used to write out the current value of the mode mask:

36252 `$ umask`
36253 `0002`

36254 (The output format is unspecified, but historical implementations use the octal integer mode
36255 format.)

36256 `$ umask -S`
36257 `u=rwx,g=rwx,o=rx`

36258 Either of these outputs can be used as the mask operand to a subsequent invocation of the *umask*
36259 utility.

36260 Assuming the mode mask is set as above, the command:

36261 `umask g-w`

36262 sets the mode mask so that subsequently created files have their S_IWGRP and S_IWOTH bits
36263 cleared.

36264 The command:

36265 `umask -- -w`

36266 sets the mode mask so that subsequently created files have all their write bits cleared. Note that
36267 *mask* operands **-r**, **-w**, **-x** or anything beginning with a hyphen, must be preceded by "--" to
36268 keep it from being interpreted as an option.

36269 **RATIONALE**

36270 Since *umask* affects the current shell execution environment, it is generally provided as a shell
36271 regular built-in. If it is called in a subshell or separate utility execution environment, such as one
36272 of the following:

36273 `(umask 002)`
36274 `nohup umask ...`
36275 `find . -exec umask ... \;`

36276 it does not affect the file mode creation mask of the environment of the caller.

36277 The description of the historical utility was modified to allow it to use the symbolic modes of
36278 *chmod*. The **-s** option used in early proposals was changed to **-S** because **-s** could be confused

with a *symbolic_mode* form of mask referring to the S_ISUID and S_ISGID bits.

The default output style is implementation-defined to permit implementors to provide migration to the new symbolic style at the time most appropriate to their users. An **–o** flag to force octal mode output was omitted because the octal mode may not be sufficient to specify all of the information that may be present in the file mode creation mask when more secure file access permission checks are implemented.

It has been suggested that trusted systems developers might appreciate ameliorating the requirement that the mode mask ‘‘affects’’ the file access permissions, since it seems access control lists might replace the mode mask to some degree. The wording has been changed to say that it affects the file permission bits, and it leaves the details of the behavior of how they affect the file access permissions to the description in the System Interfaces volume of IEEE Std 1003.1-200x.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod, the System Interfaces volume of IEEE Std 1003.1-200x, *umask*( )

CHANGE HISTORY

First released in Issue 2.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The octal mode is supported.

36301 **NAME**

36302 unalias — remove alias definitions

36303 **SYNOPSIS**

36304 UP
```
unalias alias-name...
```

36305
```
unalias –a
```

36306

36307 **DESCRIPTION**

36308 The *unalias* utility shall remove the definition for each alias name specified. See Section 2.3.1 (on
36309 page 2234). The aliases shall be removed from the current shell execution environment; see
36310 Section 2.12 (on page 2263).

36311 **OPTIONS**

36312 The *unalias* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
36313 12.2, Utility Syntax Guidelines.

36314 The following option shall be supported:

36315 **–a** Remove all alias definitions from the current shell execution environment.

36316 **OPERANDS**

36317 The following operand shall be supported:

36318 *alias-name* The name of an alias to be removed.

36319 **STDIN**

36320 Not used.

36321 **INPUT FILES**

36322 None.

36323 **ENVIRONMENT VARIABLES**

36324 The following environment variables shall affect the execution of *unalias*:

36325 *LANG* Provide a default value for the internationalization variables that are unset or null.
36326 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
36327 Internationalization Variables for the precedence of internationalization variables
36328 used to determine the values of locale categories.)

36329 *LC_ALL* If set to a non-empty string value, override the values of all the other
36330 internationalization variables.

36331 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
36332 characters (for example, single-byte as opposed to multi-byte characters in
36333 arguments).

36334 *LC_MESSAGES*
36335 Determine the locale that should be used to affect the format and contents of
36336 diagnostic messages written to standard error.

36337 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

36338 **ASYNCHRONOUS EVENTS**

36339 Default.

36340 **STDOUT**

36341 Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 One of the *alias-name* operands specified did not represent a valid alias definition, or an error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

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

EXAMPLES

None.

RATIONALE

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 this volume of IEEE Std 1003.1-200x, such as command line editing.

The **–a** option is the equivalent of the *unalias* * form of the C shell and is provided to address security concerns about unknown aliases entering the environment of a user (or application) through the allowable implementation-defined predefined alias route or as a result of an *ENV* file. (Although *unalias* could be used to simplify the ‘‘secure’’ shell script shown in the *command* rationale, it does not obviate the need to quote all command names. An initial call to *unalias* **–a** would have to be quoted in case there was an alias for *unalias*.)

FUTURE DIRECTIONS

None.

SEE ALSO

alias

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

NAME

uname — return system name

SYNOPSIS

```
uname [–snrvma]
```

DESCRIPTION

By default, the *uname* utility shall write the operating system name to standard output. When options are specified, symbols representing one or more system characteristics shall be written to the standard output. The format and contents of the symbols are implementation-defined. On systems conforming to the System Interfaces volume of IEEE Std 1003.1-200x, the symbols written shall be those supported by the *uname*( ) function as defined in the System Interfaces volume of IEEE Std 1003.1-200x.

OPTIONS

The *uname* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–a Behave as though all of the options **–mnrsv** were specified.

–m Write the name of the hardware type on which the system is running to standard output.

–n Write the name of this node within an implementation-defined communications network.

–r Write the current release level of the operating system implementation.

–s Write the name of the implementation of the operating system.

–v Write the current version level of this release of the operating system implementation.

If no options are specified, the *uname* utility shall write the operating system name, as if the **–s** option had been specified.

OPERANDS

None.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *uname*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
By default, the output shall be a single line of the following form:

`"%s\n", <sysname>`

If the –**a** option is specified, the output shall be a single line of the following form:

```
"%s %s %s %s %s\n", <sysname>, <nodename>, <release>,
    <version>, <machine>
```

Additional implementation-defined symbols may be written; all such symbols shall be written at the end of the line of output before the <newline>.

If options are specified to select different combinations of the symbols, only those symbols shall be written, in the order shown above for the –**a** option. If a symbol is not selected for writing, its corresponding trailing <blank>s also shall not be written.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 The requested information was successfully written.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
Note that any of the symbols could include embedded <space>s, which may affect parsing algorithms if multiple options are selected for output.

The node name is typically a name that the system uses to identify itself for intersystem communication addressing.

EXAMPLES
The following command:

```
uname –sr
```

writes the operating system name and release level, separated by one or more <blank>s.

RATIONALE

It was suggested that this utility cannot be used portably since the format of the symbols is implementation-defined. The POSIX.1 working group could not achieve consensus on defining these formats in the underlying *uname*( ) function, and there was no expectation that this volume of IEEE Std 1003.1-200x would be any more successful. Some applications may still find this historical utility of value. For example, the symbols could be used for system log entries or for comparison with operator or user input.

FUTURE DIRECTIONS

None.

SEE ALSO

The System Interfaces volume of IEEE Std 1003.1-200x, *uname*( )

CHANGE HISTORY

First released in Issue 2.

NAME

uncompress — expand compressed data

SYNOPSIS

XSI
```
uncompress [–cfv][file...]
```

DESCRIPTION

The *uncompress* utility shall restore files to their original state after they have been compressed using the *compress* utility. If no files are specified, the standard input shall be uncompressed to the standard output. If the invoking process has appropriate privileges, the ownership, modes, access time, and modification time of the original file shall be preserved.

This utility shall support the uncompressing of any files produced by the *compress* utility on the same implementation. For files produced by *compress* on other systems, *uncompress* supports 9 to 14-bit compression (see *compress* (on page 2465), **–b**); it is implementation-defined whether values of **–b** greater than 14 are supported.

OPTIONS

The *uncompress* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–c Write to standard output; no files are changed.

–f Do not prompt for overwriting files. Except when run in the background, if **–f** is not given the user shall be prompted as to whether an existing file should be overwritten. If the standard input is not a terminal and **–f** is not given, *uncompress* shall write a diagnostic message to standard error and exit with a status greater than zero.

–v Write messages to standard error concerning the expansion of each file.

OPERANDS

The following operand shall be supported:

file A pathname of a file. If *file* already has the **.Z** suffix specified, it shall be used as the input file and the output file shall be named **file** with the **.Z** suffix removed. Otherwise, *file* shall be used as the name of the output file and **file** with the **.Z** suffix appended shall be used as the input file.

STDIN

The standard input shall be used only if no *file* operands are specified, or if a *file* operand is `'–'`.

INPUT FILES

Input files shall be in the format produced by the *compress* utility.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *uncompress*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

When there are no *file* operands or the **–c** option is specified, the uncompressed output is written to standard output.

STDERR

Prompts shall be written to the standard error output under the conditions specified in the DESCRIPTION and OPTIONS sections. The prompts shall contain the *file* pathname, but their format is otherwise unspecified. Otherwise, the standard error output shall be used only for diagnostic messages.

OUTPUT FILES

Output files are the same as the respective input files to *compress*.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

The input file remains unmodified.

APPLICATION USAGE

The limit of 14 on the *compress* **–b** *bits* argument is to achieve portability to all systems (within the restrictions imposed by the lack of an explicit published file format). Some implementations based on 16-bit architectures cannot support 15 or 16-bit uncompression.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

compress, *zcat*

CHANGE HISTORY

First released in Issue 4.

Issue 6

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

36559 **NAME**

36560 unexpand — convert spaces to tabs

36561 **SYNOPSIS**

36562 UP
```
unexpand [ –a| –t tablist][file...]
```
36563

36564 **DESCRIPTION**

36565 The *unexpand* utility shall copy files or standard input to standard output, converting <blank>s
36566 at the beginning of each line into the maximum number of <tab>s followed by the minimum
36567 number of <space>s needed to fill the same column positions originally filled by the translated
36568 <blank>s. By default, tabstops shall be set at every eighth column position. Each <backspace>
36569 shall be copied to the output, and shall cause the column position count for tab calculations to be
36570 decremented; the count shall never be decremented to a value less than one.

36571 **OPTIONS**

36572 The *unexpand* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x,
36573 Section 12.2, Utility Syntax Guidelines.

36574 The following options shall be supported:

36575 **–a** In addition to translating <blank>s at the beginning of each line, translate all
36576 sequences of two or more <blank>s immediately preceding a tab stop to the
36577 maximum number of <tab>s followed by the minimum number of <space>s
36578 needed to fill the same column positions originally filled by the translated
36579 <blank>s.

36580 **–t** *tablist* Specify the tab stops. The application shall ensure that the *tablist* option-argument
36581 is a single argument consisting of a single positive decimal integer or multiple
36582 positive decimal integers, separated by <blank>s or commas, in ascending order. If
36583 a single number is given, tabs shall be set *tablist* column positions apart instead of
36584 the default 8. If multiple numbers are given, the tabs shall be set at those specific
36585 column positions.

36586 The application shall ensure that each tab-stop position *N* is an integer value
36587 greater than zero, and the list shall be in strictly ascending order. This is taken to
36588 mean that, from the start of a line of output, tabbing to position *N* shall cause the
36589 next character output to be in the (*N*+1)th column position on that line. When the
36590 **–t** option is not specified, the default shall be the equivalent of specifying **–t 8**
36591 (except for the interaction with **–a**, described below).

36592 No <space>-to-<tab> conversions shall occur for characters at positions beyond
36593 the last of those specified in a multiple tab-stop list.

36594 When **–t** is specified, the presence or absence of the **–a** option shall be ignored;
36595 conversion shall not be limited to the processing of leading <blank>s.

36596 **OPERANDS**

36597 The following operand shall be supported:

36598 *file* A pathname of a text file to be used as input.

36599 **STDIN**

36600 See the INPUT FILES section.

INPUT FILES

The input files shall be text files.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *unexpand*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte 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 an output device.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall be equivalent to the input files with the specified <space>-to-<tab> conversions.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

One non-intuitive aspect of *unexpand* is its restriction to leading spaces when neither **–a** nor **–t** is specified. Users who desire to always convert all spaces in a file can easily alias *unexpand* to use the **–a** or **–t 8** option.

EXAMPLES

None.

RATIONALE

On several occasions, consideration was given to adding a **–t** option to the *unexpand* utility to complement the **–t** in *expand* (see *expand* (on page 2627)). 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. An early proposal omitted **–t** because it seemed outside the scope of the UPE; it was not described in any of the base documents. However, 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 standard developers. Thus, *unexpand* is now the logical converse of *expand*.

FUTURE DIRECTIONS

None.

SEE ALSO

expand, *tabs*

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The definition of the *LC_CTYPE* environment variable is changed to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

36664 **NAME**

36665 unget — undo a previous get of an SCCS file (**DEVELOPMENT**)

36666 **SYNOPSIS**

36667 XSI
```
unget [–ns][–r SID] file...
```
36668

36669 **DESCRIPTION**

36670 The *unget* utility shall reverse the effect of a *get* **–e** done prior to creating the intended new delta.

36671 **OPTIONS**

36672 The *unget* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
36673 12.2, Utility Syntax Guidelines.

36674 The following options shall be supported:

36675 **–r** *SID* Uniquely identify which delta is no longer intended. (This would have been
36676 specified by *get* as the new delta.) The use of this option is necessary only if two or
36677 more outstanding *get* commands for editing on the same SCCS file were done by
36678 the same person (login name).

36679 **–s** Suppress the writing to standard output of the intended delta's SID.

36680 **–n** Retain the file that was obtained by *get*, which would normally be removed from
36681 the current directory.

36682 **OPERANDS**

36683 The following operands shall be supported:

36684 *file* A pathname of an existing SCCS file or a directory. If *file* is a directory, the *unget*
36685 utility shall behave as though each file in the directory were specified as a named
36686 file, except that non-SCCS files (last component of the pathname does not begin
36687 with **s.**) and unreadable files shall be silently ignored.

36688 If exactly one *file* operand appears, and it is `'–'`, the standard input shall be read;
36689 each line of the standard input shall be taken to be the name of an SCCS file to be
36690 processed. Non-SCCS files and unreadable files shall be silently ignored.

36691 **STDIN**

36692 The standard input shall be a text file used only when the *file* operand is specified as `'–'`. Each
36693 line of the text file shall be interpreted as an SCCS pathname.

36694 **INPUT FILES**

36695 Any SCCS files processed shall be files of an unspecified format.

36696 **ENVIRONMENT VARIABLES**

36697 The following environment variables shall affect the execution of *unget*:

36698 *LANG* Provide a default value for the internationalization variables that are unset or null.
36699 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
36700 Internationalization Variables for the precedence of internationalization variables
36701 used to determine the values of locale categories.)

36702 *LC_ALL* If set to a non-empty string value, override the values of all the other
36703 internationalization variables.

36704 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
36705 characters (for example, single-byte as opposed to multi-byte characters in
36706 arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
The standard output shall consist of a line for each file, in the following format:

`"%s\n", <SID removed from file>`

If there is more than one named file or if a directory or standard input is named, each pathname shall be written before each of the preceding lines:

`"\n%s:\n", <pathname>`

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
Any SCCS files updated shall be files of an unspecified format. During processing of a *file*, a locking *z-file*, as described in *get*, and a *q-file* (a working copy of the *p-file*), may be created and deleted. The *p-file* and *g-file*, as described in *get*, shall be deleted.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
None.

EXAMPLES
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
delta, *get*, *sact*

CHANGE HISTORY
First released in Issue 2.

Issue 6
The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

The normative text is reworded to emphasize the term ‘‘shall’’ for implementation requirements.

NAME

uniq — report or filter out repeated lines in a file

SYNOPSIS

```
uniq [-c|-d|-u][-f fields][-s char][input_file [output_file]]
```

DESCRIPTION

The *uniq* utility shall read an input file comparing adjacent lines, and writes one copy of each input line on the output. The second and succeeding copies of repeated adjacent input lines shall not be written.

Repeated lines in the input shall not be detected if they are not adjacent.

OPTIONS

The *uniq* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–c Precede each output line with a count of the number of times the line occurred in the input.

–d Suppress the writing of lines that are not repeated in the input.

–f *fields* Ignore the first *fields* fields on each input line when doing comparisons, where *fields* is a positive decimal integer. A field is the maximal string matched by the basic regular expression:

```
[[:blank:]]*[^[:blank:]]*
```

If the *fields* option-argument specifies more fields than appear on an input line, a null string shall be used for comparison.

–s *chars* Ignore the first *chars* characters when doing comparisons, where *chars* shall be a positive decimal integer. If specified in conjunction with the **–f** option, the first *chars* characters after the first *fields* fields shall be ignored. If the *chars* option-argument specifies more characters than remain on an input line, a null string shall be used for comparison.

–u Suppress the writing of lines that are repeated in the input.

OPERANDS

The following operands shall be supported:

input_file A pathname of the input file. If the *input_file* operand is not specified, or if the *input_file* is `'–'`, the standard input shall be used.

output_file A pathname of the output file. If the *output_file* operand is not specified, the standard output shall be used. The results are unspecified if the file named by *output_file* is the file named by *input_file*.

STDIN

The standard input shall be used only if no *input_file* operand is specified or if *input_file* is `'–'`. See the INPUT FILES section.

INPUT FILES

The input file shall be a text file.

36788 **ENVIRONMENT VARIABLES**

36789 The following environment variables shall affect the execution of *uniq*:

36790 *LANG* Provide a default value for the internationalization variables that are unset or null.
36791 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
36792 Internationalization Variables for the precedence of internationalization variables
36793 used to determine the values of locale categories.)

36794 *LC_ALL* If set to a non-empty string value, override the values of all the other
36795 internationalization variables.

36796 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
36797 characters (for example, single-byte as opposed to multi-byte characters in
36798 arguments and input files) which characters constitute a <blank> in the current
36799 locale.

36800 *LC_MESSAGES*
36801 Determine the locale that should be used to affect the format and contents of
36802 diagnostic messages written to standard error.

36803 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

36804 **ASYNCHRONOUS EVENTS**

36805 Default.

36806 **STDOUT**

36807 The standard output shall be used only if no *output_file* operand is specified. See the OUTPUT
36808 FILES section.

36809 **STDERR**

36810 The standard error shall be used only for diagnostic messages.

36811 **OUTPUT FILES**

36812 If the **–c** option is specified, the application shall ensure that the output file is empty or each line
36813 shall be of the form:

36814 `"%d %s", <number of duplicates>, <line>`

36815 otherwise, the application shall ensure that the output file is empty or each line shall be of the
36816 form:

36817 `"%s", <line>`

36818 **EXTENDED DESCRIPTION**

36819 None.

36820 **EXIT STATUS**

36821 The following exit values shall be returned:

36822 0 The utility executed successfully.

36823 >0 An error occurred.

36824 **CONSEQUENCES OF ERRORS**

36825 Default.

APPLICATION USAGE

The *sort* utility can be used to cause repeated lines to be adjacent in the input file.

EXAMPLES

The following input file data (but flushed left) was used for a test series on *uniq*:

```
#01 foo0 bar0 foo1 bar1
#02 bar0 foo1 bar1 foo1
#03 foo0 bar0 foo1 bar1
#04
#05 foo0 bar0 foo1 bar1
#06 foo0 bar0 foo1 bar1
#07 bar0 foo1 bar1 foo0
```

What follows is a series of test invocations of the *uniq* utility that use a mixture of *uniq* options against the input file data. These tests verify the meaning of *adjacent*. The *uniq* utility views the input data as a sequence of strings delimited by `'\n'`. Accordingly, for the *fields*th member of the sequence, *uniq* interprets unique or repreated adjacent lines strictly relative to the *fields*+1th member.

1. This first example tests the line counting option, comparing each line of the input file data starting from the second field:

   ```
   uniq –c –f 1 uniq_0I.t
       1 #01 foo0 bar0 foo1 bar1
       1 #02 bar0 foo1 bar1 foo0
       1 #03 foo0 bar0 foo1 bar1
       1 #04
       2 #05 foo0 bar0 foo1 bar1
       1 #07 bar0 foo1 bar1 foo0
   ```

   The number `'2'`, prefixing the fifth line of output, signifies that the *uniq* utility detected a pair of repeated lines. Given the input data, this can only be true when *uniq* is run using the **–f 1** option (which shall cause *uniq* to ignore the first field on each input line).

2. The second example tests the option to suppress unique lines, comparing each line of the input file data starting from the second field:

   ```
   uniq –d –f 1 uniq_0I.t
   #05 foo0 bar0 foo1 bar1
   ```

3. This test suppresses repeated lines, comparing each line of the input file data starting from the second field:

   ```
   uniq –u –f 1 uniq_0I.t
   #01 foo0 bar0 foo1 bar1
   #02 bar0 foo1 bar1 foo1
   #03 foo0 bar0 foo1 bar1
   #04
   #07 bar0 foo1 bar1 foo0
   ```

4. This suppresses unique lines, comparing each line of the input file data starting from the third character:

   ```
   uniq –d –s 2 uniq_0I.t
   ```

   In the last example, the *uniq* utility found no input matching the above criteria.

36870 **RATIONALE**

36871 Some historical implementations have limited lines to be 1080 bytes in length, which does not
36872 meet the implied {LINE_MAX} limit.

36873 **FUTURE DIRECTIONS**

36874 None.

36875 **SEE ALSO**

36876 *comm*, *sort*

36877 **CHANGE HISTORY**

36878 First released in Issue 2.

36879 **Issue 6**

36880 The obsolescent SYNOPSIS and associated text are removed.

36881 The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

NAME

unlink — call the *unlink*( ) function

SYNOPSIS

XSI
```
unlink file
```

DESCRIPTION

The *unlink* utility shall perform the function call:

```
unlink(file);
```

A user may need appropriate privilege to invoke the *unlink* utility.

OPTIONS

None.

OPERANDS

The following operands shall be supported:

file The pathname of an existing file.

STDIN

Not used.

INPUT FILES

Not used.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *unlink*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

None.

STDERR

The standard error shall be used only for diagnostic messages.

36921 **OUTPUT FILES**

36922 None.

36923 **EXTENDED DESCRIPTION**

36924 None.

36925 **EXIT STATUS**

36926 The following exit values shall be returned:

36927 0 Successful completion.

36928 >0 An error occurred.

36929 **CONSEQUENCES OF ERRORS**

36930 Default.

36931 **APPLICATION USAGE**

36932 None.

36933 **EXAMPLES**

36934 None.

36935 **RATIONALE**

36936 None.

36937 **FUTURE DIRECTIONS**

36938 None.

36939 **SEE ALSO**

36940 *link*, *rm*, the System Interfaces volume of IEEE Std 1003.1-200x, *unlink*( )

36941 **CHANGE HISTORY**

36942 First released in Issue 5.

NAME

uucp — system-to-system copy

SYNOPSIS

XSI
```
uucp [–cCdfjmr][–n user] source-file... destination-file
```

DESCRIPTION

The *uucp* utility shall copy files named by the *source-file* arguments to the *destination-file* argument. The files named can be on local or remote systems.

The *uucp* utility cannot guarantee support for all character encodings in all circumstances. For example, transmission data may be restricted to 7 bits by the underlying network, 8-bit data and filenames need not be portable to non-internationalized systems, and so on. Under these circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used, and that only characters defined in the portable filename character set be used for naming files. The protocol for transfer of files is unspecified by IEEE Std 1003.1-200x.

Typical implementations of this utility require a communications line configured to use the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface, but other communications means may be used. On systems where there are no available communications means (either temporarily or permanently), this utility shall write an error message describing the problem and exit with a non-zero exit status.

OPTIONS

The *uucp* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–c Do not copy local file to the spool directory for transfer to the remote machine (default).

–C Force the copy of local files to the spool directory for transfer.

–d Make all necessary directories for the file copy (default).

–f Do not make intermediate directories for the file copy.

–j Write the job identification string to standard output. This job identification can be used by *uustat* to obtain the status or terminate a job.

–m Send mail to the requester when the copy is completed.

–n *user* Notify *user* on the remote system that a file was sent.

–r Do not start the file transfer; just queue the job.

OPERANDS

The following operands shall be supported:

destination-file, *source-file*
A pathname of a file to be copied to, or from, respectively. Either name can be a pathname on the local machine, or can have the form:

```
system-name!pathname
```

where *system-name* is taken from a list of system names that *uucp* knows about. The destination *system-name* can also be a list of names such as:

36985 *system-name*!*system-name*!...!*system-name*!*pathname*

36986 in which case, an attempt is made to send the file via the specified route to the
36987 destination. Care should be taken to ensure that intermediate nodes in the route
36988 are willing to forward information.

36989 The shell pattern matching notation characters '?', '*', and "[...]" appearing
36990 in *pathname* shall be expanded on the appropriate system.

36991 Pathnames can be one of:

36992 1. An absolute pathname.

36993 2. A pathname preceded by ~*user* where *user* is a login name on the specified
36994 system and is replaced by that user's login directory. Note that if an invalid
36995 login is specified, the default is to the public directory (called *PUBDIR*; the
36996 actual location of *PUBDIR* is implementation-defined).

36997 3. A pathname preceded by ~/*destination* where *destination* is appended to
36998 *PUBDIR*.

36999 **Note:** This destination is treated as a filename unless more than one file is being
37000 transferred by this request or the destination is already a directory. To
37001 ensure that it is a directory, follow the destination with a '/'. For
37002 example, **~/dan/** as the destination makes the directory **PUBDIR/dan** if it
37003 does not exist and put the requested files in that directory.

37004 4. Anything else shall be prefixed by the current directory.

37005 If the result is an erroneous pathname for the remote system, the copy shall fail. If
37006 the *destination-file* is a directory, the last part of the *source-file* name shall be used.

37007 The read, write, and execute permissions given by *uucp* are implementation-
37008 defined.

37009 **STDIN**

37010 Not used.

37011 **INPUT FILES**

37012 The files to be copied are regular files.

37013 **ENVIRONMENT VARIABLES**

37014 The following environment variables shall affect the execution of *uucp*:

37015 *LANG* Provide a default value for the internationalization variables that are unset or null.
37016 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
37017 Internationalization Variables for the precedence of internationalization variables
37018 used to determine the values of locale categories.)

37019 *LC_ALL* If set to a non-empty string value, override the values of all the other
37020 internationalization variables.

37021 *LC_COLLATE*
37022 Determine the locale for the behavior of ranges, equivalence classes and multi-
37023 character collating elements within bracketed filename patterns.

37024 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
37025 characters (for example, single-byte as opposed to multi-byte characters in
37026 arguments and input files) and the behavior of character classes within bracketed
37027 filename patterns (for example, "'[[:lower:]]*'").

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error, and informative messages written to standard output.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
Not used.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
The output files (which may be on other systems) are copies of the input files.

If the **–m** is used, mail files are modified.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
The domain of remotely accessible files can (and for obvious security reasons usually should) be severely restricted.

Note that the '!' character in addresses has to be escaped when using *csh* as a command interpreter because of its history substitution syntax. For *ksh* and *sh* the escape is not necessary, but may be used.

As noted above, shell metacharacters appearing in pathnames are expanded on the appropriate system. On an internationalized system, this is done under the control of local settings of *LC_COLLATE* and *LC_CTYPE*. Thus, care should be taken when using bracketed filename patterns, as collation and typing rules may vary from one system to another. Also be aware that certain types of expression (that is, equivalence classes, character classes, and collating symbols) need not be supported on non-internationalized systems.

EXAMPLES
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO

mailx, *uuencode*, *uustat*, *uux*

CHANGE HISTORY

First released in Issue 2.

Issue 6

The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section. |

The UN margin codes and associated shading are removed from the **–C**, **–f**, **–j**, **–n**, and **–r** options in response to The Open Group Base Resolution bwg2001-003. | |

NAME

uudecode — decode a binary file

SYNOPSIS

UP

```
uudecode [–o outfile][file]
```

DESCRIPTION

The *uudecode* utility shall read a file, or standard input if no file is specified, that includes data created by the *uuencode* utility. The *uudecode* utility shall scan the input file, searching for data compatible with one of the formats specified in *uuencode* and attempt to create or overwrite the file described by the data (or overridden by the **–o** option). The pathname shall be contained in the data or specified by the **–o** option. The file access permission bits and contents for the file to be produced shall be contained in that data. The mode bits of the created file (other than standard output) shall be set from the file access permission bits contained in the data; that is, other attributes of the mode, including the file mode creation mask (see *umask*), shall not affect the file being produced.

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.

OPTIONS

The *uudecode* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following option shall be supported by the implementation:

–o *outfile* A pathname of a file that shall be used instead of any pathname contained in the input data. Specifying an *outfile* option-argument of **/dev/stdout** shall indicate standard output.

OPERANDS

The following operand shall be supported:

file The pathname of a file containing the output of *uuencode*.

STDIN

See the INPUT FILES section.

INPUT FILES

The input files shall be files containing the output of *uuencode*.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *uudecode*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
If the file data header encoded by *uuencode* is – or **/dev/stdout**, or the **–o /dev/stdout** option overrides the file data, the standard output shall be in the same format as the file originally encoded by *uuencode*. Otherwise, the standard output shall not be used.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
The output file shall be in the same format as the file originally encoded by *uuencode*.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE
The user who is invoking *uudecode* must have write permission on any file being created.

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.

EXAMPLES
None.

RATIONALE
Input files are not necessarily text files, as stated by an early proposal. 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 **–o** option is not historical practice, but was added at the request of WG15 so that the user could override the target pathname without having to edit the input data itself.

In early drafts, the [**–o** *outfile*] option-argument allowed the use of – to mean standard output. The symbol – has only been used previously in IEEE Std 1003.1-200x as a standard input indicator. The developers of the standard did not wish to overload the meaning of – in this manner. The **/dev/stdout** concept exists on most modern systems. The **/dev/stdout** syntax does not refer to a new special file. It is just a magic cookie to specify standard output.

FUTURE DIRECTIONS

None.

SEE ALSO

uuencode

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The **–o** *outfile* option is added, as specified in the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

37172 **NAME**

37173 uuencode — encode a binary file

37174 **SYNOPSIS**

37175 UP

```
uuencode [–m][file] decode_pathname
```

37176

37177 **DESCRIPTION**

37178 The *uuencode* utility shall write an encoded version of the named input file, or standard input if
37179 no *file* is specified, to standard output. The output shall be encoded using one of the algorithms
37180 described in the STDOUT section and shall include the file access permission bits (in *chmod* octal
37181 or symbolic notation) of the input file and the *decode_pathname*, for re-creation of the file on
37182 another system that conforms to this volume of IEEE Std 1003.1-200x.

37183 **OPTIONS**

37184 The *uuencode* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x,
37185 Section 12.2, Utility Syntax Guidelines.

37186 The following option shall be supported by the implementation:

37187 **–m** Encode the output using the MIME Base64 algorithm described in STDOUT. If **–m**
37188 is not specified, the historical algorithm described in STDOUT shall be used.

37189 **OPERANDS**

37190 The following operands shall be supported:

37191 *decode_pathname*
37192 The pathname of the file into which the *uudecode* utility shall place the decoded
37193 file. Specifying a *decode_pathname* operand of **/dev/stdout** shall indicate that
37194 *uudecode* is to use standard output. If there are characters in *decode_pathname* that
37195 are not in the portable filename character set the results are unspecified.

37196 *file* A pathname of the file to be encoded.

37197 **STDIN**

37198 See the INPUT FILES section.

37199 **INPUT FILES**

37200 Input files can be files of any type.

37201 **ENVIRONMENT VARIABLES**

37202 The following environment variables shall affect the execution of *uuencode*:

37203 *LANG* Provide a default value for the internationalization variables that are unset or null.
37204 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
37205 Internationalization Variables for the precedence of internationalization variables
37206 used to determine the values of locale categories.)

37207 *LC_ALL* If set to a non-empty string value, override the values of all the other
37208 internationalization variables.

37209 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
37210 characters (for example, single-byte as opposed to multi-byte characters in
37211 arguments and input files).

37212 *LC_MESSAGES*
37213 Determine the locale that should be used to affect the format and contents of
37214 diagnostic messages written to standard error.

37215 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

37216 **ASYNCHRONOUS EVENTS**

37217 Default.

37218 **STDOUT**

37219 **uuencode Base64 Algorithm**

37220 The standard output shall be a text file (encoded in the character set of the current locale) that
37221 begins with the line:

37222 `"begin-base64Δ%sΔ%s\n", <mode>, <decode_pathname>`

37223 and ends with the line:

37224 `"====\n"`

37225 In both cases, the lines shall have no preceding or trailing <blank>s.

37226 The encoding process represents 24-bit groups of input bits as output strings of four encoded
37227 characters. Proceeding from left to right, a 24-bit input group shall be formed by concatenating
37228 three 8-bit input groups. Each 24-bit input group then shall be treated as four concatenated 6-bit
37229 groups, each of which shall be translated into a single digit in the base64 alphabet. When
37230 encoding a bit stream via the base64 encoding, the bit stream shall be presumed to be ordered
37231 with the most-significant bit first. That is, the first bit in the stream shall be the high-order bit in
37232 the first byte, and the eighth bit shall be the low-order bit in the first byte, and so on. Each 6-bit
37233 group is used as an index into an array of 64 printable characters, as shown in Table 4-21.

37234 **Table 4-21** uuencode Base64 Values

37235

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
37236 0	A	17	R	34	i	51	z
37237 1	B	18	S	35	j	52	0
37238 2	C	19	T	36	k	53	1
37239 3	D	20	U	37	l	54	2
37240 4	E	21	V	38	m	55	3
37241 5	F	22	W	39	n	56	4
37242 6	G	23	X	40	o	57	5
37243 7	H	24	Y	41	p	58	6
37244 8	I	25	Z	42	q	59	7
37245 9	J	26	a	43	r	60	8
37246 10	K	27	b	44	s	61	9
37247 11	L	28	c	45	t	62	+
37248 12	M	29	d	46	u	63	/
37249 13	N	30	e	47	v		
37250 14	O	31	f	48	w	(pad)	=
37251 15	P	32	g	49	x		
37252 16	Q	33	h	50	y		

37253 The character referenced by the index shall be placed in the output string.

37254 The output stream (encoded bytes) shall be represented in lines of no more than 76 characters
37255 each. All line breaks or other characters not found in the table shall be ignored by decoding
37256 software (see *uudecode*).

37257 Special processing shall be performed if fewer than 24 bits are available at the end of a message
37258 or encapsulated part of a message. A full encoding quantum shall always be completed at the

37259 end of a message. When fewer than 24 input bits are available in an input group, zero bits shall
37260 be added (on the right) to form an integral number of 6-bit groups. Output character positions
37261 that are not required to represent actual input data shall be set to the character `'='`. Since all
37262 base64 input is an integral number of octets, only the following cases can arise:

37263 1. The final quantum of encoding input is an integral multiple of 24 bits; here, the final unit of
37264 encoded output shall be an integral multiple of 4 characters with no `'='` padding.

37265 2. The final quantum of encoding input is exactly 16 bits; here, the final unit of encoded
37266 output shall be three characters followed by one `'='` padding character.

37267 3. The final quantum of encoding input is exactly 8 bits; here, the final unit of encoded output
37268 shall be two characters followed by two `'='` padding characters.

37269 A terminating `"===="` evaluates to nothing and denotes the end of the encoded data.

37270 **uuencode Historical Algorithm**

37271 The standard output shall be a text file (encoded in the character set of the current locale) that
37272 begins with the line:

37273 `"begin`Δ`%s`Δ`%s\n"` *<mode>, <decode_pathname>*

37274 and ends with the line:

37275 `"end\n"`

37276 In both cases, the lines shall have no preceding or trailing <blank>s.

37277 The algorithm that shall be used for lines in between **begin** and **end** takes three octets as input
37278 and writes four characters of output by splitting the input at six-bit intervals into four octets,
37279 containing data in the lower six bits only. These octets shall be converted to characters by adding
37280 a value of 0x20 to each octet, so that each octet is in the range [0x20,0x5f], and then it shall be |
37281 assumed to represent a printable character in the ISO/IEC 646:1991 standard encoded character |
37282 set. It then shall be translated into the corresponding character codes for the codeset in use in the
37283 current locale. (For example, the octet 0x41, representing `'A'`, would be translated to `'A'` in the
37284 current codeset, such as 0xc1 if it were EBCDIC.)

37285 Where the bits of two octets are combined, the least significant bits of the first octet shall be
37286 shifted left and combined with the most significant bits of the second octet shifted right. Thus
37287 the three octets *A*, *B*, *C* shall be converted into the four octets:

```
37288 0x20 + (( A >> 2                    ) & 0x3F)
37289 0x20 + (((A << 4) | ((B >> 4) & 0xF)) & 0x3F)
37290 0x20 + (((B << 2) | ((C >> 6) & 0x3)) & 0x3F)
37291 0x20 + (( C                         ) & 0x3F)
```

37292 These octets then shall be translated into the local character set.

37293 Each encoded line contains a length character, equal to the number of characters to be decoded
37294 plus 0x20 translated to the local character set as described above, followed by the encoded
37295 characters. The maximum number of octets to be encoded on each line shall be 45.

37296 **STDERR**

37297 The standard error shall be used only for diagnostic messages. |

37298 **OUTPUT FILES**

37299 None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

The file is expanded by 35 percent (each three octets become four, plus control information) causing it to take longer to transmit.

Since this utility is intended to create files to be used for data interchange between systems with possibly different codesets, and to represent binary data as a text file, the ISO/IEC 646:1991 standard 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 646:1991 standard codeset, 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 codeset, 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 codeset, 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 codeset. This translation only makes sense from the local codeset, not if the file has been put into a ISO/IEC 646:1991 standard representation first. Similarly, files processed by *uuencode* can be placed in *pax* archives, intermixed with other text files in the same codeset.

EXAMPLES

None.

RATIONALE

A new algorithm was added at the request of the international community to parallel work in RFC 2045 (MIME). As with the historical *uuencode* format, the Base64 Content-Transfer-Encoding is designed to represent arbitrary sequences of octets in a form that is not humanly readable. A 65-character subset of the ISO/IEC 646:1991 standard is used, enabling 6 bits to be represented per printable character. (The extra 65th character, `'='`, is used to signify a special processing function.)

This subset has the important property that it is represented identically in all versions of the ISO/IEC 646:1991 standard, including US ASCII, and all characters in the subset are also represented identically in all versions of EBCDIC. The historical *uuencode* algorithm does not share this property, which is the reason that a second algorithm was added to the ISO POSIX-2 standard.

The string `"===="` was used for the termination instead of the end used in the original format because the latter is a string that could be valid encoded input.

In an early draft, the **–m** option was named **–b** (for Base64), but it was renamed to reflect its relationship to the RFC 2045. A **–u** was also present to invoke the default algorithm, but since this was not historical practice, it was omitted as being unnecessary.

See the RATIONALE section in *uudecode* for the derivation of the **/dev/stdout** symbol.

FUTURE DIRECTIONS

None.

SEE ALSO

mailx, *uudecode*

CHANGE HISTORY

First released in Issue 4.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The Base64 algorithm and the ability to output to **/dev/stdout** are added as specified in the IEEE P1003.2b draft standard.

NAME

uustat — uucp status inquiry and job control

SYNOPSIS

XSI

```
uustat [ –q| –k jobid| –r jobid]
uustat [–s system][–u user]
```

DESCRIPTION

The *uustat* utility shall display the status of, or cancel, previously specified *uucp* requests, or provide general status on *uucp* connections to other systems.

When no options are given, *uustat* shall write to standard output the status of all *uucp* requests issued by the current user.

Typical implementations of this utility require a communications line configured to use the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface, but other communications means may be used. On systems where there are no available communications means (either temporarily or permanently), this utility shall write an error message describing the problem and exit with a non-zero exit status.

OPTIONS

The *uustat* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–q Write the jobs queued for each machine.

–k *jobid* Kill the *uucp* request whose job identification is *jobid*. The application shall ensure that the killed *uucp* request belongs to the person invoking *uustat* unless that user has appropriate privileges.

–r *jobid* Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup program from deleting the job until the jobs modification time reaches the limit imposed by the program.

–s *system* Write the status of all *uucp* requests for remote system *system*.

–u *user* Write the status of all *uucp* requests issued by *user*.

OPERANDS

None.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *uustat*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error, and informative messages written to standard output.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall consist of information about each job selected, in an unspecified format. The information shall include at least the job ID, the user ID or name, and the remote system name.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

uucp

CHANGE HISTORY

First released in Issue 2.

Issue 6

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

The *LC_TIME* and *TZ* entries are removed from the ENVIRONMENT VARIABLES section.

The UN margin code and associated shading are removed from the **−q** option in reaponse to The Open Group Base Resolution bwg2001-003.

NAME

uux — remote command execution

SYNOPSIS

XSI
```
uux [–np] command–string
uux [–jnp] command–string
```

DESCRIPTION

The *uux* utility shall gather zero or more files from various systems, execute a shell pipeline (see Section 2.9 (on page 2248)) on a specified system, and then send the standard output of the command to a file on a specified system. Only the first command of a pipeline can have a *system-name*! prefix. All other commands in the pipeline shall be executed on the system of the first command.

The following restrictions are applicable to the shell pipeline processed by *uux*:

- In gathering files from different systems, pathname expansion shall not be performed by *uux*. Thus, a request such as:

  ```
  uux "c99 remsys!~/*.c"
  ```

  would attempt to copy the file named literally ***.c** to the local system.

- The redirection operators ">>", "<<", ">|", and ">&" shall not be accepted. Any use of these redirection operators shall cause this utility to write an error message describing the problem and exit with a non-zero exit status.

- The reserved word **!** cannot be used at the head of the pipeline to modify the exit status. (See the *command-string* operand description below.)

- Alias substitution shall not be performed.

A filename can be specified as for *uucp*; it can be an absolute pathname, a pathname preceded by *~name* (which is replaced by the corresponding login directory), a pathname specified as *~/dest* (*dest* is prefixed by the public directory called *PUBDIR*; the actual location of *PUBDIR* is implementation-defined), or a simple filename (which is prefixed by *uux* with the current directory). See *uucp* (on page 3163) for the details.

The execution of commands on remote systems shall take place in an execution directory known to the *uucp* system. All files required for the execution shall be put into this directory unless they already reside on that machine. Therefore, the application shall ensure that non-local filenames (without path or machine reference) are unique within the *uux* request.

The *uux* utility shall attempt to get all files to the execution system. For files that are output files, the application shall ensure that the filename is escaped using parentheses.

The remote system shall notify the user by mail if the requested command on the remote system was disallowed or the files were not accessible. This notification can be turned off by the **–n** option.

Typical implementations of this utility require a communications line configured to use the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General Terminal Interface, but other communications means may be used. On systems where there are no available communications means (either temporarily or permanently), this utility shall write an error message describing the problem and exits with a non-zero exit status.

The *uux* utility cannot guarantee support for all character encodings in all circumstances. For example, transmission data may be restricted to 7 bits by the underlying network, 8-bit data and

filenames need not be portable to non-internationalized systems, and so on. Under these circumstances, it is recommended that only characters defined in the ISO/IEC 646:1991 standard International Reference Version (equivalent to ASCII) 7-bit range of characters be used and that only characters defined in the portable filename character set be used for naming files.

OPTIONS

The *uux* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–p Make the standard input to *uux* the standard input to the *command-string*.

–j Write the job identification string to standard output. This job identification can be used by *uustat* to obtain the status or terminate a job.

–n Do not notify the user if the command fails.

OPERANDS

The following operand shall be supported:

command-string
A string made up of one or more arguments that are similar to normal command arguments, except that the command and any filenames can be prefixed by *system-name*!. A null *system-name* shall be interpreted as the local system.

STDIN

The standard input shall not be used unless the '–' or **–p** option is specified; in those cases, the standard input shall be made the standard input of the *command-string*.

INPUT FILES

Input files shall be selected according to the contents of *command-string*.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *uux*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall be not used unless the –**j** option is specified; in that case, the job identification string shall be written to standard output in the following format:

```
"%s\n", <jobid>
```

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

Output files shall be created or written, or both, according to the contents of *command-string*.

If the –**n** is not used, mail files shall be modified following any command or file-access failures on the remote system.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

Note that, for security reasons, many installations limit the list of commands executable on behalf of an incoming request from *uux*. Many sites permit little more than the receipt of mail via *uux*.

Any characters special to the command interpreter should be quoted either by quoting the entire *command-string* or quoting the special characters as individual arguments.

As noted in *uucp*, shell pattern matching notation characters appearing in pathnames are expanded on the appropriate local system. This is done under the control of local settings of *LC_COLLATE* and *LC_CTYPE*. Thus, care should be taken when using bracketed filename patterns, as collation and typing rules may vary from one system to another. Also be aware that certain types of expression (that is, equivalence classes, character classes, and collating symbols) need not be supported on non-internationalized systems.

EXAMPLES

1. The following command gets **file1** from system **a** and **file2** file from system **b**, executes *diff* on the local system, and puts the results in **file.diff** in the local *PUBDIR* directory. (*PUBDIR* is the *uucp* public directory on the local system.)

   ```
   uux "!diff a!/usr/file1 b!/a4/file2 >!~/file.diff"
   ```

2. The following command fails because *uux* places all files copied to a system in the same working directory. Although the files **xyz** are from two different systems, their filenames are the same and conflict.

   ```
   uux "!diff a!/usr1/xyz b!/usr2/xyz >!~/xyz.diff"
   ```

3. The following command succeeds (assuming *diff* is permitted on system **a**) because the file local to system **a** is not copied to the working directory, and hence does not conflict the file from system **c**.

```
uux "a!diff a!/usr/xyz c!/usr/xyz >!~/xyz.diff"
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

uucp, *uuencode*, *uustat*

CHANGE HISTORY

First released in Issue 2.

Issue 6

The obsolescent SYNOPSIS is removed.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements. |

The UN margin code and associated shading are removed from the **–j** option in response to The Open Group Base Resolution bwg2001-003. | |

37581 **NAME**

37582 val — validate SCCS files (**DEVELOPMENT**)

37583 **SYNOPSIS**

37584 XSI
```
val –
```

37585
```
val [–s][–m name][–r SID][–y type] file...
```

37586

37587 **DESCRIPTION**

37588 The *val* utility shall determine whether the specified *file* is an SCCS file meeting the
37589 characteristics specified by the options.

37590 **OPTIONS**

37591 The *val* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2,
37592 Utility Syntax Guidelines, except that the usage of the '–' operand is not strictly as intended by
37593 the guidelines (that is, reading options and operands from standard input).

37594 The following options shall be supported:

37595 **–m** *name* Specify a *name*, which is compared with the SCCS %**M**% keyword in *file*; see *get*
37596 (on page 2675).

37597 **–r** *SID* Specify a *SID* (SCCS Identification String), an SCCS delta number. A check shall be
37598 made to determine whether the *SID* is ambiguous (for example, **–r 1** is ambiguous
37599 because it physically does not exist but implies 1.1, 1.2, and so on, which may
37600 exist) or invalid (for example, **–r 1.0** or **–r 1.1.0** are invalid because neither case can
37601 exist as a valid delta number). If the *SID* is valid and not ambiguous, a check shall
37602 be made to determine whether it actually exists.

37603 **–s** Silence the diagnostic message normally written to standard output for any error
37604 that is detected while processing each named file on a given command line.

37605 **–y** *type* Specify a *type*, which shall be compared with the SCCS %**Y**% keyword in *file*; see
37606 *get* (on page 2675).

37607 **OPERANDS**

37608 The following operands shall be supported:

37609 *file* A pathname of an existing SCCS file. If exactly one *file* operand appears, and it is
37610 '–', the standard input shall be read: each line shall be independently processed |
37611 as if it were a command line argument list. (However, the line is not subjected to |
37612 any of the shell word expansions, such as parameter expansion or quote removal.) |

37613 **STDIN**

37614 The standard input shall be a text file used only when the *file* operand is specified as '–'.

37615 **INPUT FILES**

37616 Any SCCS files processed shall be files of an unspecified format. |

37617 **ENVIRONMENT VARIABLES**

37618 The following environment variables shall affect the execution of *val*:

37619 *LANG* Provide a default value for the internationalization variables that are unset or null.
37620 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
37621 Internationalization Variables for the precedence of internationalization variables
37622 used to determine the values of locale categories.)

37623 *LC_ALL* If set to a non-empty string value, override the values of all the other
37624 internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error, and informative messages written to standard output.

NLSPATH Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

The standard output shall consist of informative messages about either:

1. Each file processed
2. Each command line read from standard input

If the standard input is not used, for each *file* operand yielding a discrepancy, the output line shall have the following format:

`"%s: %s\n", <pathname>, <unspecified string>`

If standard input is used, a line of input shall be written before each of the preceding lines for files containing discrepancies:

`"%s:\n", <input line>`

STDERR

Not used.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The 8-bit code returned by *val* shall be a disjunction of the possible errors, that is, it can be interpreted as a bit string where set bits are interpreted as follows:

0x80	=	Missing file argument.
0x40	=	Unknown or duplicate option.
0x20	=	Corrupted SCCS file.
0x10	=	Cannot open file or file not SCCS.
0x08	=	*SID* is invalid or ambiguous.
0x04	=	*SID* does not exist.
0x02	=	%**Y**%, –**y** mismatch.
0x01	=	%**M**%, –**m** mismatch.

Note that *val* can process two or more files on a given command line and can process multiple command lines (when reading the standard input). In these cases an aggregate code shall be returned: a logical OR of the codes generated for each command line and file processed.

37665 **CONSEQUENCES OF ERRORS**

37666 Default.

37667 **APPLICATION USAGE**

37668 Since the *val* exit status sets the 0x80 bit, shell applications checking "$?" cannot tell if it
37669 terminated due to a missing file argument or receipt of a signal.

37670 **EXAMPLES**

37671 In a directory with three SCCS files, **s.x** (of **t** type ‘‘text’’), **s.y**, and **s.z** (a corrupted file), the
37672 following command could produce the output shown:

```
37673     val – <<EOF
37674     –y source s.x
37675     –m y s.y
37676     s.z
37677     EOF
37678     –y source s.x
37679         s.x: %Y%, –y mismatch
37680     s.z
37681         s.z: corrupted SCCS file
```

37682 **RATIONALE**

37683 None.

37684 **FUTURE DIRECTIONS**

37685 None.

37686 **SEE ALSO**

37687 *admin*, *delta*, *get*, *prs*

37688 **CHANGE HISTORY**

37689 First released in Issue 2.

37690 **Issue 6**

37691 The Open Group Corrigendum U025/4 is applied, correcting a typographical error in the EXIT
37692 STATUS.

37693 The normative text is reworded to emphasize the term ‘‘shall’’ for implementation requirements.

NAME

vi — screen-oriented (visual) display editor

SYNOPSIS

UP

```
vi [–rR][–l][–c command][–t tagstring][–w size][file ...]
```

DESCRIPTION

This utility shall be provided on systems that both support the User Portability Utilities option and define the POSIX2_CHAR_TERM symbol. On other systems it is optional.

The *vi* (visual) utility is a screen-oriented text editor. Only the open and visual modes of the editor are described in IEEE Std 1003.1-200x; see the line editor *ex* for additional editing capabilities used in *vi*. The user can switch back and forth between *vi* and *ex* and execute *ex* commands from within *vi*.

This reference page uses the term *edit buffer* to describe the current working text. No specific implementation is implied by this term. All editing changes are performed on the edit buffer, and no changes to it shall affect any file until an editor command writes the file.

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 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 volume of IEEE Std 1003.1-200x or report an error describing the terminal-related deficiency.

OPTIONS

The *vi* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

–c *command* See the *ex* command description of the **–c** option.

–r See the *ex* command description of the **–r** option.

–R See the *ex* command description of the **–R** option.

–t *tagstring* See the *ex* command description of the **–t** option.

–w *size* See the *ex* command description of the **–w** option.

OPERANDS

See the OPERANDS section of the *ex* command for a description of the operands supported by the *vi* command.

STDIN

If standard input is not a terminal device, the results are undefined. The standard input consists of a series of commands and input text, as described in the EXTENDED DESCRIPTION section.

If a read from the standard input returns an error, or if the editor detects an end-of-file condition from the standard input, it shall be equivalent to a SIGHUP asynchronous event.

INPUT FILES

See the INPUT FILES section of the *ex* command for a description of the input files supported by the *vi* command.

ENVIRONMENT VARIABLES

See the ENVIRONMENT VARIABLES section of the *ex* command for the environment variables that affect the execution of the *vi* command.

ASYNCHRONOUS EVENTS

See the ASYNCHRONOUS EVENTS section of the *ex* for the asynchronous events that affect the execution of the *vi* command.

STDOUT

If standard output is not a terminal device, undefined results occur.

Standard output may be used for writing prompts to the user, for informational messages, and for writing lines from the file.

STDERR

If standard output is not a terminal device, undefined results occur.

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

See the OUTPUT FILES section of the *ex* command for a description of the output files supported by the *vi* command.

EXTENDED DESCRIPTION

If the terminal does not have the capabilities necessary to support an unspecified portion of the *vi* definition, implementations shall start initially in *ex* mode or open mode. Otherwise, after initialization, *vi* shall be in command mode; text input mode can be entered by one of several commands used to insert or change text. In text input mode, <ESC> can be used to return to command mode; other uses of <ESC> are described later in this section; see **Terminate Command or Input Mode** (on page 3194).

Initialization in ex and vi

See **Initialization in ex and vi** (on page 2559) for a description of *ex* and *vi* initialization for the *vi* utility.

Command Descriptions in vi

The following symbols are used in this reference page to represent arguments to commands.

buffer See the description of *buffer* in the EXTENDED DESCRIPTION section of the *ex* utility; see **Command Descriptions in ex** (on page 2569).

In open and visual mode, when a command synopsis shows both [*buffer*] and [*count*] preceding the command name, they can be specified in either order.

count A positive integer used as an optional argument to most commands, either to give a repeat count or as a size. This argument is optional and shall default to 1 unless otherwise specified.

The Synopsis lines for the *vi* commands <control>-G, <control>-L, <control>-R, <control>-], **%**, **&**, **^**, **D**, **m**, **M**, **Q**, **u**, **U**, and **ZZ** do not have *count* as an optional argument. Regardless, it shall not be an error to specify a *count* to these commands, and any specified *count* shall be ignored.

motion An optional trailing argument used by the **!**, **<**, **>**, **c**, **d**, and **y** commands, which is used to indicate the region of text that shall be affected by the command. The motion can be either one of the command characters repeated or one of several other *vi* commands (listed in the following table). Each of the applicable commands specifies the region of text matched by repeating the command; each command that can be used as a motion command specifies the region of text it affects.

Commands that take *motion* arguments operate on either lines or characters, depending on the circumstances. When operating on lines, all lines that fall partially or wholly within the text region specified for the command shall be affected. When operating on characters, only the exact characters in the specified text region shall be affected. Each motion command specifies this individually.

When commands that may be motion commands are not used as motion commands, they shall set the current position to the current line and column as specified.

The following commands shall be valid cursor motion commands:

<control>-H	(	E	]]	t
<newline>	)	F	^	w
<carriage-return>	+	G	b	{
<control>-N	,	H	e	|
<control>-P	-	L	f	}
<space>	/	M	h	0
$	_	N	j	
%	;	T	k	
'character	?	W	l	
‘character	B	[[	n	

Any *count* that is specified to a command that has an associated motion command shall be applied to the motion command. If a *count* is applied to both the command and its associated motion command, the effect shall be multiplicative.

The following symbols are used in this section to specify locations in the edit buffer:

current character
The character that is currently indicated by the cursor.

end of a line
The point located between the last non-<newline> (if any) and the terminating <newline> of a line. For an empty line, this location coincides with the beginning of the line.

end of the edit buffer
The location corresponding to the end of the last line in the edit buffer.

The following symbols are used in this section to specify command actions:

bigword In the POSIX locale, *vi* shall recognize four kinds of *bigwords*:

1. A maximal sequence of non-<blank>s preceded and followed by <blank>s or the beginning or end of a line or the edit buffer
2. One or more sequential blank lines
3. The first character in the edit buffer
4. The last non-<newline> in the edit buffer

word In the POSIX locale, *vi* shall recognize five kinds of words:

1. A maximal sequence of letters, digits, and underscores, delimited at both ends by:
   - — Characters other than letters, digits, or underscores
   - — The beginning or end of a line
   - — The beginning or end of the edit buffer
2. A maximal sequence of characters other than letters, digits, underscores, or <blank>s, delimited at both ends by:
   - — A letter, digit, underscore
   - — <blank>s
   - — The beginning or end of a line
   - — The beginning or end of the edit buffer
3. One or more sequential blank lines
4. The first character in the edit buffer
5. The last non-<newline> in the edit buffer

section boundary

A *section boundary* is one of the following:

1. A line whose first character is a <form-feed>
2. A line whose first character is an open curly brace (`'{'`)
3. A line whose first character is a period and whose second and third characters match a two-character pair in the **sections** edit option (see *ed*)
4. A line whose first character is a period and whose only other character matches the first character of a two-character pair in the **sections** edit option, where the second character of the two-character pair is a <space>
5. The first line of the edit buffer
6. The last line of the edit buffer if the last line of the edit buffer is empty or if it is a **]]** or **}** command; otherwise, the last non-<newline> of the last line of the edit buffer

paragraph boundary

A *paragraph boundary* is one of the following:

1. A section boundary
2. A line whose first character is a period and whose second and third characters match a two-character pair in the **paragraphs** edit option (see *ed*)
3. A line whose first character is a period and whose only other character matches the first character of a two-character pair in the *paragraphs* edit option, where the second character of the two-character pair is a <space>
4. One or more sequential blank lines

remembered search direction

See the description of remembered search direction in *ed*.

37858 *sentence boundary*

37859 A *sentence boundary* is one of the following:

37860 1. A paragraph boundary

37861 2. The first non-<blank> that occurs after a paragraph boundary

37862 3. The first non-<blank> that occurs after a period (`'.'`), exclamation mark (`'!'`),
37863 or question mark (`'?'`), followed by two <space>s or the end of a line; any
37864 number of closing parenthesis (`')'`), closing brackets (`']'`), double quote (`'"'`),
37865 or single quote (`'\''`) characters can appear between the punctuation mark and
37866 the two <space>s or end-of-line

37867 In the remainder of the description of the *vi* utility, the term ''buffer line'' refers to a line in the
37868 edit buffer and the term ''display line'' refers to the line or lines on the display screen used to
37869 display one buffer line. The term ''current line'' refers to a specific ''buffer line''.

37870 If there are display lines on the screen for which there are no corresponding buffer lines because
37871 they correspond to lines that would be after the end of the file, they shall be displayed as a single
37872 tilde (`'~'`) character, plus the terminating <newline>.

37873 The last line of the screen shall be used to report errors or display informational messages. It
37874 shall also be used to display the input for ''line-oriented commands'' (**/**, **?**, **:**, and **!**). When a line-
37875 oriented command is executed, the editor shall enter text input mode on the last line on the
37876 screen, using the respective command characters as prompt characters. (In the case of the **!**
37877 command, the associated motion shall be entered by the user before the editor enters text input
37878 mode.) The line entered by the user shall be terminated by a <newline>, a non-<control>-V-
37879 escaped <carriage-return>, or unescaped <ESC>. It is unspecified if more characters than
37880 require a display width minus one column number of screen columns can be entered.

37881 If any command is executed that overwrites a portion of the screen other than the last line of the
37882 screen (for example, the *ex* **suspend**, or **!** commands), other than the *ex* **shell** command, the user
37883 shall be prompted for a character before the screen is refreshed and the edit session continued.

37884 <tab>s shall take up the number of columns on the screen set by the **tabstop** edit option (see *ed*),
37885 unless there are less than that number of columns before the display margin that will cause the
37886 displayed line to be folded; in this case, they shall only take up the number of columns up to that
37887 boundary.

37888 The cursor shall be placed on the current line and relative to the current column as specified by
37889 each command described in the following sections.

37890 In open mode, if the current line is not already displayed, then it shall be displayed.

37891 In visual mode, if the current line is not displayed, then the lines that are displayed shall be
37892 expanded, scrolled, or redrawn to cause an unspecified portion of the current line to be
37893 displayed. If the screen is redrawn, no more than the number of display lines specified by the
37894 value of the **window** edit option shall be displayed (unless the current line cannot be completely
37895 displayed in the number of display lines specified by the **window** edit option) and the current
37896 line shall be positioned as close to the center of the displayed lines as possible (within the
37897 constraints imposed by the distance of the line from the beginning or end of the edit buffer). If
37898 the current line is before the first line in the display and the screen is scrolled, an unspecified
37899 portion of the current line shall be placed on the first line of the display. If the current line is after
37900 the last line in the display and the screen is scrolled, an unspecified portion of the current line
37901 shall be placed on the last line of the display.

37902 In visual mode, if a line from the edit buffer (other than the current line) does not entirely fit into
37903 the lines at the bottom of the display that are available for its presentation, the editor may

choose not to display any portion of the line. The lines of the display that do not contain text from the edit buffer for this reason shall each consist of a single '@' character.

In visual mode, the editor may choose for unspecified reasons to not update lines in the display to correspond to the underlying edit buffer text. The lines of the display that do not correctly correspond to text from the edit buffer for this reason shall consist of a single '@' character (plus the terminating <newline>), and the <control>-R command shall cause the editor to update the screen to correctly represent the edit buffer.

Open and visual mode commands that set the current column set it to a column position in the display, and not a character position in the line. In this case, however, the column position in the display shall be calculated for a infinite width display; for example, the column related to a character that is part of a line that has been folded onto additional screen lines will be offset from the display line column where the buffer line begins, not from the beginning of a particular display line.

The display cursor column in the display is based on the value of the current column, as follows, with each rule applied in turn:

1. If the current column is after the last display line column used by the displayed line, the display cursor column shall be set to the last display line column occupied by the last non-<newline> in the current line; otherwise, the display cursor column shall be set to the current column.
2. If the character of which some portion is displayed in the display line column specified by the display cursor column requires more than a single display line column:
    a. If in text input mode, the display cursor column shall be adjusted to the first display line column in which any portion of that character is displayed.
    b. Otherwise, the display cursor column shall be adjusted to the last display line column in which any portion of that character is displayed.

The current column shall not be changed by these adjustments to the display cursor column.

If an error occurs during the parsing or execution of a *vi* command:

- The terminal shall be alerted. Execution of the *vi* command shall stop, and the cursor (for example, the current line and column) shall not be further modified.
- Unless otherwise specified by the following command sections, it is unspecified whether an informational message shall be displayed.
- Any partially entered *vi* command shall be discarded.
- If the *vi* command resulted from a **map** expansion, all characters from that **map** expansion shall be discarded, except as otherwise specified by the **map** command (see *ed*).
- If the *vi* command resulted from the execution of a buffer, no further commands caused by the execution of the buffer shall be executed.

37940 **Page Backwards**

37941 *Synopsis*: `[`*count*`]` <control>-B

37942 If in open mode, the <control>-B command shall behave identically to the **z** command.
37943 Otherwise, if the current line is the first line of the edit buffer, it shall be an error.

37944 If the **window** edit option is less than 3, display a screen where the last line of the display shall
37945 be some portion of:

37946 (*current first line*) –1

37947 otherwise, display a screen where the first line of the display shall be some portion of:

37948 (*current first line*) – *count* x ((window edit option) –2)

37949 If this calculation would result in a line that is before the first line of the edit buffer, the first line
37950 of the display shall display some portion of the first line of the edit buffer.

37951 *Current line*: If no lines from the previous display remain on the screen, set to the last line of the
37952 display; otherwise, set to (*line* – the number of new lines displayed on this screen).

37953 *Current column*: Set to non-<blank>.

37954 **Scroll Forward**

37955 *Synopsis*: `[`*count*`]` <control>-D

37956 If the current line is the last line of the edit buffer, it shall be an error.

37957 If no *count* is specified, *count* shall default to the *count* associated with the previous <control>-D
37958 or <control>-U command. If there was no previous <control>-D or <control>-U command, *count*
37959 shall default to the value of the **scroll** edit option.

37960 If in open mode, write lines starting with the line after the current line, until *count* lines or the
37961 last line of the file have been written.

37962 *Current line*: If the current line + *count* is past the last line of the edit buffer, set to the last line of
37963 the edit buffer; otherwise, set to the current line + *count*.

37964 *Current column*: Set to non-<blank>.

37965 **Scroll Forward by Line**

37966 *Synopsis*: `[`*count*`]` <control>-E

37967 Display the line count lines after the last line currently displayed.

37968 If the last line of the edit buffer is displayed, it shall be an error. If there is no line *count* lines
37969 after the last line currently displayed, the last line of the display shall display some portion of
37970 the last line of the edit buffer.

37971 *Current line*: Unchanged if the previous current character is displayed; otherwise, set to the first
37972 line displayed.

37973 *Current column*: Unchanged.

37974 **Page Forward**

37975 *Synopsis*: **[***count***]** <control>-F

37976 If in open mode, the <control>-F command shall behave identically to the **z** command.
37977 Otherwise, if the current line is the last line of the edit buffer, it shall be an error.

37978 If the **window** edit option is less than 3, display a screen where the first line of the display shall
37979 be some portion of:

37980 (*current last line*) +1

37981 otherwise, display a screen where the first line of the display shall be some portion of:

37982 (*current first line*) + *count* x ((window edit option) –2)

37983 If this calculation would result in a line that is after the last line of the edit buffer, the last line of
37984 the display shall display some portion of the last line of the edit buffer.

37985 *Current line*: If no lines from the previous display remain on the screen, set to the first line of the
37986 display; otherwise, set to (*line* + the number of new lines displayed on this screen).

37987 *Current column*: Set to non-<blank>.

37988 **Display Information**

37989 *Synopsis*: <control>-G

37990 This command shall be equivalent to the *ex* **file** command .

37991 **Move Cursor Backwards**

37992 *Synopsis*: **[***count***]** <control>-H
37993 **[***count***]** h
37994 the current *erase* character (see stty)

37995 If there are no characters before the current character on the current line, it shall be an error. If
37996 there are less than *count* previous characters on the current line, *count* shall be adjusted to the
37997 number of previous characters on the line.

37998 If used as a motion command:

37999 1. The text region shall be from the character before the starting cursor up to and including
38000 the *count*th character before the starting cursor.

38001 2. Any text copied to a buffer shall be in character mode.

38002 If not used as a motion command:

38003 *Current line*: Unchanged.

38004 *Current column*: Set to (*column* – the number of columns occupied by *count* characters ending
38005 with the previous current column).

38006 **Move Down**

38007 *Synopsis*: `[`*count*`]` <newline>
38008 `[`*count*`]` <control>-J
38009 `[`*count*`]` <control>-M
38010 `[`*count*`]` <control>-N
38011 `[`*count*`]` j
38012 `[`*count*`]` <carriage-return>
38013 `[`*count*`]` +

38014 If there are less than *count* lines after the current line in the edit buffer, it shall be an error.

38015 If used as a motion command:

38016 1. The text region shall include the starting line and the next *count* – 1 lines.

38017 2. Any text copied to a buffer shall be in line mode.

38018 If not used as a motion command:

38019 *Current line*: Set to *current line*+ *count*.

38020 *Current column*: Set to non-<blank> for the <carriage-return>, <control>-M, and **+** commands;
38021 otherwise, unchanged.

38022 **Clear and Redisplay**

38023 *Synopsis*: <control>-L

38024 If in open mode, clear the screen and redisplay the current line. Otherwise, clear and redisplay
38025 the screen.

38026 *Current line*: Unchanged.

38027 *Current column*: Unchanged.

38028 **Move Up**

38029 *Synopsis*: `[`*count*`]` <control>-P
38030 `[`*count*`]` k
38031 `[`*count*`]` –

38032 If there are less than *count* lines before the current line in the edit buffer, it shall be an error.

38033 If used as a motion command:

38034 1. The text region shall include the starting line and the previous *count* lines.

38035 2. Any text copied to a buffer shall be in line mode.

38036 If not used as a motion command:

38037 *Current line*: Set to *current line* – *count*.

38038 *Current column*: Set to non-<blank> for the – command; otherwise, unchanged.

38039 **Redraw Screen**

38040 *Synopsis*: `<control>-R`

38041 If any lines have been deleted from the display screen and flagged as deleted on the terminal
38042 using the **@** convention (see the beginning of the EXTENDED DESCRIPTION section), they shall
38043 be redisplayed to match the contents of the edit buffer.

38044 It is unspecified whether lines flagged with **@** because they do not fit on the terminal display
38045 shall be affected.

38046 *Current line*: Unchanged.

38047 *Current column*: Unchanged.

38048 **Scroll Backward**

38049 *Synopsis*: `[`*count*`] <control>-U`

38050 If the current line is the first line of the edit buffer, it shall be an error.

38051 If no *count* is specified, *count* shall default to the *count* associated with the previous <control>-D
38052 or <control>-U command. If there was no previous <control>-D or <control>-U command, *count*
38053 shall default to the value of the **scroll** edit option.

38054 *Current line*: If *count* is greater than the current line, set to 1; otherwise, set to the current line –
38055 *count*.

38056 *Current column*: Set to non-<blank>.

38057 **Scroll Backward by Line**

38058 *Synopsis*: `[`*count*`] <control>-Y`

38059 Display the line *count* lines before the first line currently displayed.

38060 If the current line is the first line of the edit buffer, it shall be an error. If this calculation would
38061 result in a line that is before the first line of the edit buffer, the first line of the display shall
38062 display some portion of the first line of the edit buffer.

38063 *Current line*: Unchanged if the previous current character is displayed; otherwise, set to the first
38064 line displayed.

38065 *Current column*: Unchanged.

38066 **Edit the Alternate File**

38067 *Synopsis*: `<control>-^`

38068 This command shall be equivalent to the *ex* **edit** command, with the alternate pathname as its
38069 argument.

38070 **Terminate Command or Input Mode**

38071 *Synopsis*: `<ESC>`

38072 If a partial *vi* command (as defined by at least one, non-*count* character) has been entered,
38073 discard the *count* and the command character(s).

38074 Otherwise, if no command characters have been entered, and the <ESC> was the result of a map
38075 expansion, the terminal shall be alerted and the <ESC> character shall be discarded, but it shall
38076 not be an error.

Otherwise, it shall be an error.

Current line: Unchanged.

Current column: Unchanged.

Search for tagstring

Synopsis: `<control>-]`

If the current character is not a word or <blank>, it shall be an error.

This command shall be equivalent to the *ex* **tag** command, with the argument to that command defined as follows.

If the current character is a <blank>:

1. Skip all <blank>s after the cursor up to the end of the line.
2. If the end of the line is reached, it shall be an error.

Then, the argument to the *ex* **tag** command shall be the current character and all subsequent characters, up to the first non-word character or the end of the line.

Move Cursor Forward

Synopsis: `[`*count*`]` `<space>`
`[`*count*`]` `l` (ell)

If there are less than *count* non-<newline>s after the cursor on the current line, *count* shall be adjusted to the number of non-<newline>s after the cursor on the line.

If used as a motion command:

1. If the current or *count*th character after the cursor is the last non-<newline> in the line, the text region shall be comprised of the current character up to and including the last non-<newline> in the line. Otherwise, the text region shall be from the current character up to, but not including, the *count*th character after the cursor.
2. Any text copied to a buffer shall be in character mode.

If not used as a motion command:

If there are no non-<newline>s after the current character on the current line, it shall be an error.

Current line: Unchanged.

Current column: Set to the last column that displays any portion of the *count*th character after the current character.

Replace Text with Results from Shell Command

Synopsis: `[`*count*`]` `!` *motion shell-commands* `<newline>`

If the motion command is the **!** command repeated:

1. If the edit buffer is empty and no *count* was supplied, the command shall be the equivalent of the *ex* **:read !** command, with the text input, and no text shall be copied to any buffer.
2. Otherwise:
   a. If there are less than *count* –1 lines after the current line in the edit buffer, it shall be an error.

b. The text region shall be from the current line up to and including the next *count* –1 lines.

Otherwise, the text region shall be the lines in which any character of the text region specified by the motion command appear.

Any text copied to a buffer shall be in line mode.

This command shall be equivalent to the *ex* **!** command for the specified lines.

Move Cursor to End-of-line

Synopsis: **[***count***]** **$**

It shall be an error if there are less than (*count* –1) lines after the current line in the edit buffer.

If used as a motion command:

1. If *count* is 1:
   a. It shall be an error if the line is empty.
   b. Otherwise, the text region shall consist of all characters from the starting cursor to the last non-<newline> in the line, inclusive, and any text copied to a buffer shall be in character mode.
2. Otherwise, if the starting cursor position is at or before the first non-<blank> in the line, the text region shall consist of the current and the next *count* –1 lines, and any text saved to a buffer shall be in line mode.
3. Otherwise, the text region shall consist of all characters from the starting cursor to the last non-<newline> in the line that is *count* –1 lines forward from the current line, and any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Set to the *current line* + *count*–1.

Current column: The current column is set to the last display line column of the last non-<newline> in the line, or column position 1 if the line is empty.

The current column shall be adjusted to be on the last display line column of the last non-<newline> of the current line as subsequent commands change the current line, until a command changes the current column.

Move to Matching Character

Synopsis: **%**

If the character at the current position is not a parenthesis, bracket, or curly brace, search forward in the line to the first one of those characters. If no such character is found, it shall be an error.

The matching character shall be the parenthesis, bracket, or curly brace matching the parenthesis, bracket, or curly brace, respectively, that was at the current position or that was found on the current line.

Matching shall be determined as follows, for an open parenthesis:

1. Set a counter to 1.
2. Search forwards until a parenthesis is found or the end of the edit buffer is reached.

3. If the end of the edit buffer is reached, it shall be an error.
4. If an open parenthesis is found, increment the counter by 1.
5. If a close parenthesis is found, decrement the counter by 1.
6. If the counter is zero, the current character is the matching character.

Matching for a close parenthesis shall be equivalent, except that the search shall be backwards, from the starting character to the beginning of the buffer, a close parenthesis shall increment the counter by 1, and an open parenthesis shall decrement the counter by 1.

Matching for brackets and curly braces shall be equivalent, except that searching shall be done for open and close brackets or open and close curly braces. It is implementation-defined whether other characters are searched for and matched as well.

If used as a motion command:

1. If the matching cursor was after the starting cursor in the edit buffer, and the starting cursor position was at or before the first non-<blank> non-<newline> in the starting line, and the matching cursor position was at or after the last non-<blank> non-<newline> in the matching line, the text region shall consist of the current line to the matching line, inclusive, and any text copied to a buffer shall be in line mode.
2. If the matching cursor was before the starting cursor in the edit buffer, and the starting cursor position was at or after the last non-<blank> non-<newline> in the starting line, and the matching cursor position was at or before the first non-<blank> non-<newline> in the matching line, the text region shall consist of the current line to the matching line, inclusive, and any text copied to a buffer shall be in line mode.
3. Otherwise, the text region shall consist of the starting character to the matching character, inclusive, and any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Set to the line where the matching character is located.

Current column: Set to the last column where any portion of the matching character is displayed.

Repeat Substitution

Synopsis: &

Repeat the previous substitution command. This command shall be equivalent to the *ex* **&** command with the current line as its addresses, and without *options*, *count*, or *flags*.

Return to Previous Context at Beginning of Line

Synopsis: `' character`

It shall be an error if there is no line in the edit buffer marked by *character*.

If used as a motion command:

1. If the starting cursor is after the marked cursor, then the locations of the starting cursor and the marked cursor in the edit buffer shall be logically swapped.
2. The text region shall consist of the starting line up to and including the marked line, and any text copied to a buffer shall be in line mode.

If not used as a motion command:

Current line: Set to the line referenced by the mark.

Current column: Set to non-<blank>.

Return to Previous Context

Synopsis: `' character`

It shall be an error if the marked line is no longer in the edit buffer. If the marked line no longer contains a character in the saved numbered character position, it shall be as if the marked position is the first non-<blank>.

If used as a motion command:

1. It shall be an error if the marked cursor references the same character in the edit buffer as the starting cursor.
2. If the starting cursor is after the marked cursor, then the locations of the starting cursor and the marked cursor in the edit buffer shall be logically swapped.
3. If the starting line is empty or the starting cursor is at or before the first non-<blank> non-<newline> of the starting line, and the marked cursor line is empty or the marked cursor references the first character of the marked cursor line, the text region shall consist of all lines containing characters from the starting cursor to the line before the marked cursor line, inclusive, and any text copied to a buffer shall be in line mode.
4. Otherwise, if the marked cursor line is empty or the marked cursor references a character at or before the first non-<blank> non-<newline> of the marked cursor line, the region of text shall be from the starting cursor to the last non-<newline> of the line before the marked cursor line, inclusive, and any text copied to a buffer shall be in character mode.
5. Otherwise, the region of text shall be from the starting cursor (inclusive), to the marked cursor (exclusive), and any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Set to the line referenced by the mark.

Current column: Set to the last column in which any portion of the character referenced by the mark is displayed.

Return to Previous Section

Synopsis: `[[`

Move the cursor backward through the edit buffer to the first character of the previous section boundary, *count* times.

If used as a motion command:

1. If the starting cursor was at the first character of the starting line or the starting line was empty, and the first character of the boundary was the first character of the boundary line, the text region shall consist of the current line up to and including the line where the *count*th next boundary starts, and any text copied to a buffer shall be in line mode.
2. If the boundary was the last line of the edit buffer or the last non-<newline> of the last line of the edit buffer, the text region shall consist of the last character in the edit buffer up to and including the starting character, and any text saved to a buffer shall be in character mode.

38232 3. Otherwise, the text region shall consist of the starting character up to but not including the
38233 first character in the *count*th next boundary, and any text copied to a buffer shall be in
38234 character mode.

38235 If not used as a motion command:

38236 *Current line*: Set to the line where the *count*th next boundary in the edit buffer starts.

38237 *Current column*: Set to the last column in which any portion of the first character of the *count*th
38238 next boundary is displayed, or column position 1 if the line is empty.

38239 **Move to Next Section**

38240 *Synopsis*:]]

38241 Move the cursor forward through the edit buffer to the first character of the next section
38242 boundary, *count* times.

38243 If used as a motion command:

38244 1. If the starting cursor was at the first character of the starting line or the starting line was
38245 empty, and the first character of the boundary was the first character of the boundary line,
38246 the text region shall consist of the current line up to and including the line where the
38247 *count*th previous boundary starts, and any text copied to a buffer shall be in line mode.

38248 2. If the boundary was the first line of the edit buffer, the text region shall consist of the first
38249 character in the edit buffer up to but not including the starting character, and any text
38250 copied to a buffer shall be in character mode.

38251 3. Otherwise, the text region shall consist of the first character in the *count*th previous section
38252 boundary up to but not including the starting character, and any text copied to a buffer
38253 shall be in character mode.

38254 If not used as a motion command:

38255 *Current line*: Set to the line where the *count*th previous boundary in the edit buffer starts.

38256 *Current column*: Set to the last column in which any portion of the first character of the *count*th
38257 previous boundary is displayed, or column position 1 if the line is empty.

38258 **Move to First Non-<blank> Position on Current Line**

38259 *Synopsis*: ^

38260 If used as a motion command:

38261 1. If the line has no non-<blank> non-<newline>s, or if the cursor is at the first non-<blank>
38262 non-<newline> of the line, it shall be an error.

38263 2. If the cursor is before the first non-<blank> non-<newline> of the line, the text region shall
38264 be comprised of the current character, up to, but not including, the first non-<blank> non-
38265 <newline> of the line.

38266 3. If the cursor is after the first non-<blank> non-<newline> of the line, the text region shall
38267 be from the character before the starting cursor up to and including the first non-<blank>
38268 non-<newline> of the line.

38269 4. Any text copied to a buffer shall be in character mode.

38270 If not used as a motion command:

Current line: Unchanged.

Current column: Set to non-<blank>.

Current and line above

Synopsis: **[***count***]** _

If there are less than *count* –1 lines after the current line in the edit buffer, it shall be an error.

If used as a motion command:

1. If *count* is less than 2, the text region shall be the current line.
2. Otherwise, the text region shall include the starting line and the next *count* –1 lines.
3. Any text copied to a buffer shall be in line mode.

If not used as a motion command:

Current line: Set to current line + *count* –1.

Current column: Set to non-<blank>.

Move Back to Beginning of Sentence

Synopsis: **[***count***]** (

Move backward to the beginning of a sentence. This command shall be equivalent to the **[[** command, with the exception that sentence boundaries shall be used instead of section boundaries.

Move Forward to Beginning of Sentence

Synopsis: **[***count***]**)

Move forward to the beginning of a sentence. This command shall be equivalent to the **]]** command, with the exception that sentence boundaries shall be used instead of section boundaries.

Move Back to Preceding Paragraph

Synopsis: **[***count***]** {

Move back to the beginning of the preceding paragraph. This command shall be equivalent to the **[[** command, with the exception that paragraph boundaries shall be used instead of section boundaries.

Move Forward to Next Paragraph

Synopsis: **[***count***]** }

Move forward to the beginning of the next paragraph. This command shall be equivalent to the **]]** command, with the exception that paragraph boundaries shall be used instead of section boundaries.

38303 **Move to Specific Column Position**

38304 *Synopsis*: `[count] |`

38305 For the purposes of this command, lines that are too long for the current display and that have
38306 been folded shall be treated as having a single, 1–based, number of columns.

38307 If there are less than *count* columns in which characters from the current line are displayed on
38308 the screen, *count* shall be adjusted to be the last column in which any portion of the line is
38309 displayed on the screen.

38310 If used as a motion command:

38311 1. If the line is empty, or the cursor character is the same as the character on the *count*th
38312 column of the line, it shall be an error.

38313 2. If the cursor is before the *count*th column of the line, the text region shall be comprised of
38314 the current character, up to but not including the character on the *count*th column of the
38315 line.

38316 3. If the cursor is after the *count*th column of the line, the text region shall be from the
38317 character before the starting cursor up to and including the character on the *count*th
38318 column of the line.

38319 4. Any text copied to a buffer shall be in character mode.

38320 If not used as a motion command:

38321 *Current line*: Unchanged.

38322 *Current column*: Set to the last column in which any portion of the character that is displayed in
38323 the *count* column of the line is displayed.

38324 **Reverse Find Character**

38325 *Synopsis*: `[count] ,`

38326 If the last **F**, **f**, **T**, or **t** command was **F**, **f**, **T**, or **t**, this command shall be equivalent to an **f**, **F**, **t**, or
38327 **T** command, respectively, with the specified *count* and the same search character.

38328 If there was no previous **F**, **f**, **T**, or **t** command, it shall be an error.

38329 **Repeat**

38330 *Synopsis*: `[count] .`

38331 Repeat the last **!**, **<**, **>**, **A**, **C**, **D**, **I**, **J**, **O**, **P**, **R**, **S**, **X**, **Y**, **a**, **c**, **d**, **i**, **o**, **p**, **r**, **s**, **x**, **y**, or **~** command. It shall
38332 be an error if none of these commands have been executed. Commands (other than commands
38333 that enter text input mode) executed as a result of map expansions, shall not change the value of
38334 the last repeatable command.

38335 Repeated commands with associated motion commands shall repeat the motion command as
38336 well; however, any specified *count* shall replace the *count*(s) that were originally specified to the
38337 repeated command or its associated motion command.

38338 If the motion component of the repeated command is **f**, **F**, **t**, or **T**, the repeated command shall
38339 not set the remembered search character for the **;** and **,** commands.

38340 If the repeated command is **p** or **P**, and the buffer associated with that command was a numeric
38341 buffer named with a number less than 9, the buffer associated with the repeated command shall
38342 be set to be the buffer named by the name of the previous buffer logically incremented by 1.

If the repeated character is a text input command, the input text associated with that command is repeated literally:

- Input characters are neither macro or abbreviation-expanded.
- Input characters are not interpreted in any special way with the exception that <newline>, <carriage-return>, and <control>-T behave as described in **Input Mode Commands in vi** (on page 3220).

Current line: Set as described for the repeated command.

Current column: Set as described for the repeated command.

Find Regular Expression

Synopsis: /

If the input line contains no non-<newline>s, it shall be equivalent to a line containing only the last regular expression encountered. The enhanced regular expressions supported by *vi* are described in **Regular Expressions in ex** (on page 2592).

Otherwise, the line shall be interpreted as one or more regular expressions, optionally followed by an address offset or a *vi* **z** command.

If the regular expression is not the last regular expression on the line, or if a line offset or **z** command is specified, the regular expression shall be terminated by an unescaped '/' character, which shall not be used as part of the regular expression. If the regular expression is not the first regular expression on the line, it shall be preceded by zero or more <blank>s, a semicolon, zero or more <blank>s, a leading '/' character, which shall not be interpreted as part of the regular expression. It shall be an error to precede any regular expression with any characters other than these.

Each search shall begin from the character after the first character of the last match (or, if it is the first search, after the cursor). If the **wrapscan** edit option is set, the search shall continue to the character before the starting cursor character; otherwise, to the end of the edit buffer. It shall be an error if any search fails to find a match, and an informational message to this effect shall be displayed.

An optional address offset (see **Addressing in ex** (on page 2562)) can be specified after the last regular expression by including a trailing '/' character after the regular expression and specifying the address offset. This offset will be from the line containing the match for the last regular expression specified. It shall be an error if the line offset would indicate a line address less than 1 or greater than the last line in the edit buffer. An address offset of zero shall be supported. It shall be an error to follow the address offset with any other characters than <blank>s.

If not used as a motion command, an optional **z** command (see **Redraw Window** (on page 3219)) can be specified after the last regular expression by including a trailing '/' character after the regular expression, zero or more <blank>s, a 'z', zero or more <blank>s, an optional new **window** edit option value, zero or more <blank>s, and a location character. The effect shall be as if the **z** command was executed after the / command. It shall be an error to follow the **z** command with any other characters than <blank>s.

The remembered search direction shall be set to forward.

If used as a motion command:

1. It shall be an error if the last match references the same character in the edit buffer as the starting cursor.

2. If any address offset is specified, the last match shall be adjusted by the specified offset as described previously.
3. If the starting cursor is after the last match, then the locations of the starting cursor and the last match in the edit buffer shall be logically swapped.
4. If any address offset is specified, the text region shall consist of all lines containing characters from the starting cursor to the last match line, inclusive, and any text copied to a buffer shall be in line mode.
5. Otherwise, if the starting line is empty or the starting cursor is at or before the first non-<blank> non-<newline> of the starting line, and the last match line is empty or the last match starts at the first character of the last match line, the text region shall consist of all lines containing characters from the starting cursor to the line before the last match line, inclusive, and any text copied to a buffer shall be in line mode.
6. Otherwise, if the last match line is empty or the last match begins at a character at or before the first non-<blank> non-<newline> of the last match line, the region of text shall be from the current cursor to the last non-<newline> of the line before the last match line, inclusive, and any text copied to a buffer shall be in character mode.
7. Otherwise, the region of text shall be from the current cursor (inclusive), to the first character of the last match (exclusive), and any text copied to a buffer shall be be in character mode.

If not used as a motion command:

Current line: If a match is found, set to the last matched line plus the address offset, if any; otherwise, unchanged.

Current column: Set to the last column on which any portion of the first character in the last matched string is displayed, if a match is found; otherwise, unchanged.

Move to First Character in Line

Synopsis: `0` (zero)

Move to the first character on the current line. The character `'0'` shall not be interpreted as a command if it is immediately preceded by a digit.

If used as a motion command:

1. If the cursor character is the first character in the line, it shall be an error.
2. The text region shall be from the character before the cursor character up to and including the first character in the line.
3. Any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Unchanged.

Current column: The last column in which any portion of the first character in the line is displayed, or if the line is empty, unchanged.

38424 **Execute an ex Command**

38425 *Synopsis*: :

38426 Execute one or more *ex* commands.

38427 If any portion of the screen other than the last line of the screen was overwritten by any *ex*
38428 command (except **shell**), *vi* shall display a message indicating that it is waiting for an input from
38429 the user, and shall then read a character. This action may also be taken for other, unspecified
38430 reasons.

38431 If the next character entered is a ':', another *ex* command shall be accepted and executed. Any
38432 other character shall cause the screen to be refreshed and *vi* shall return to command mode.

38433 *Current line*: As specified for the *ex* command.

38434 *Current column*: As specified for the *ex* command.

38435 **Repeat Find**

38436 *Synopsis*: [*count*] ;

38437 This command shall be equivalent to the last **F**, **f**, **T**, or **t** command, with the specified *count*, and
38438 with the same search character used for the last **F**, **f**, **T**, or **t** command. If there was no previous **F**,
38439 **f**, **T**, or **t** command, it shall be an error.

38440 **Shift Left**

38441 *Synopsis*: [*count*] < *motion*

38442 If the motion command is the < command repeated:

38443 1. If there are less than *count* –1 lines after the current line in the edit buffer, it shall be an
38444 error.

38445 2. The text region shall be from the current line, up to and including the next *count* –1 lines.

38446 Shift any line in the text region specified by the *count* and motion command one shiftwidth (see
38447 the *ex* **shiftwidth** option) toward the start of the line, as described by the *ex* < command. The
38448 unshifted lines shall be copied to the unnamed buffer in line mode.

38449 *Current line*: If the motion was from the current cursor position toward the end of the edit
38450 buffer, unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region
38451 specified by the motion command.

38452 *Current column*: Set to non-<blank>.

38453 **Shift Right**

38454 *Synopsis*: [*count*] > *motion*

38455 If the motion command is the > command repeated:

38456 1. If there are less than *count* –1 lines after the current line in the edit buffer, it shall be an
38457 error.

38458 2. The text region shall be from the current line, up to and including the next *count* –1 lines.

38459 Shift any line with characters in the text region specified by the *count* and motion command one
38460 shiftwidth (see the *ex* **shiftwidth** option) away from the start of the line, as described by the *ex* >
38461 command. The unshifted lines shall be copied into the unnamed buffer in line mode.

Current line: If the motion was from the current cursor position toward the end of the edit buffer, unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region specified by the motion command.

Current column: Set to non-<blank>.

Scan Backwards for Regular Expression

Synopsis: `?`

Scan backwards; The **?** command shall be equivalent to the **/** command (see **Find Regular Expression** (on page 3202)) with the following exceptions:

1. The input prompt shall be a `'?'`.
2. Each search shall begin from the character before the first character of the last match (or, if it is the first search, the character before the cursor character).
3. The search direction shall be from the cursor toward the beginning of the edit buffer, and the **wrapscan** edit option shall affect whether the search wraps to the end of the edit buffer and continues.
4. The remembered search direction shall be set to backward.

Execute

Synopsis: `@`*`buffer`*

If the *buffer* is specified as **@**, the last buffer executed shall be used. If no previous buffer has been executed, it shall be an error.

Behave as if the contents of the named buffer were entered as standard input. After each line of a line-mode buffer, and all but the last line of a character mode buffer, behave as if a <newline> were entered as standard input.

If an error occurs during this process, an error message shall be written, and no more characters resulting from the execution of this command shall be processed.

If a *count* is specified, behave as if that count were entered as user input before the characters from the **@** buffer were entered.

Current line: As specified for the individual commands.

Current column: As specified for the individual commands.

Reverse Case

Synopsis: `[`*`count`*`]` `~`

Reverse the case of the current character and the next *count* –1 characters, such that lowercase characters that have uppercase counterparts shall be changed to uppercase characters, and uppercase characters that have lowercase counterparts shall be changed to lowercase characters, as prescribed by the current locale. No other characters shall be affected by this command.

If there are less than *count* –1 characters after the cursor in the edit buffer, *count* shall be adjusted to the number of characters after the cursor in the edit buffer minus 1.

For the purposes of this command, the next character after the last non-<newline> on the line shall be the next character in the edit buffer.

Current line: Set to the line including the (*count*–1)th character after the cursor.

Current column: Set to the last column in which any portion of the (*count*–1)th character after the cursor is displayed.

Append

Synopsis: **[***count***]** a

Enter text input mode after the current cursor position. No characters already in the edit buffer shall be affected by this command. A *count* shall cause the input text to be appended *count* –1 more times to the end of the input.

Current line/column: As specified for the text input commands (see **Input Mode Commands in vi** (on page 3220)).

Append at End-of-Line

Synopsis: **[***count***]** A

This command shall be equivalent to the *vi* command:

$ **[** *count* **]** a

(see **Append**).

Move Backward to Preceding Word

Synopsis: **[***count***]** b

With the exception that words are used as the delimiter instead of bigwords, this command shall be equivalent to the **B** command.

Move Backward to Preceding Bigword

Synopsis: **[***count***]** B

If the edit buffer is empty or the cursor is on the first character of the edit buffer, it shall be an error. If less than *count* bigwords begin between the cursor and the start of the edit buffer, *count* shall be adjusted to the number of bigword beginnings between the cursor and the start of the edit buffer.

If used as a motion command:

1. The text region shall be from the first character of the *count*th previous bigword beginning up to but not including the cursor character.
2. Any text copied to a buffer shall be in character mode.

If not used as a motion command:

Current line: Set to the line containing the *current column*.

Current column: Set to the last column upon which any part of the first character of the *count*th previous bigword is displayed.

38533 **Change**

38534 *Synopsis*: **[***buffer***][***count***]** c *motion*

38535 If the motion command is the **c** command repeated:

38536 1. The buffer text shall be in line mode.

38537 2. If there are less than *count* –1 lines after the current line in the edit buffer, it shall be an
38538 error.

38539 3. The text region shall be from the current line up to and including the next *count* –1 lines.

38540 Otherwise, the buffer text mode and text region shall be as specified by the motion command.

38541 The replaced text shall be copied into *buffer*, if specified, and into the unnamed buffer. If the text
38542 to be replaced contains characters from more than a single line, or the buffer text is in line mode,
38543 the replaced text shall be copied into the numeric buffers as well.

38544 If the buffer text is in line mode:

38545 1. Any lines that contain characters in the region shall be deleted, and the editor shall enter
38546 text input mode at the beginning of a new line which shall replace the first line deleted.

38547 2. If the **autoindent** edit option is set, **autoindent** characters equal to the **autoindent**
38548 characters on the first line deleted shall be inserted as if entered by the user.

38549 Otherwise, if characters from more than one line are in the region of text:

38550 1. The text shall be deleted.

38551 2. Any text remaining in the last line in the text region shall be appended to the first line in
38552 the region, and the last line in the region shall be deleted.

38553 3. The editor shall enter text input mode after the last character not deleted from the first line
38554 in the text region, if any; otherwise, on the first column of the first line in the region.

38555 Otherwise:

38556 1. If the glyph for `'$'` is smaller than the region, the end of the region shall be marked with a
38557 `'$'`.

38558 2. The editor shall enter text input mode, overwriting the region of text.

38559 *Current line/column*: As specified for the text input commands (see **Input Mode Commands in vi**
38560 (on page 3220)).

38561 **Change to End-of-Line**

38562 *Synopsis*: **[***buffer***][***count***]** C

38563 This command shall be equivalent to the *vi* command:

38564 **[***buffer***][***count***]** c$

38565 See the **c** command.

38566 **Delete**

38567 *Synopsis*: **[***buffer***][***count***]** d *motion*

38568 If the motion command is the **d** command repeated:

38569 1. The buffer text shall be in line mode.

38570 2. If there are less than *count* –1 lines after the current line in the edit buffer, it shall be an
38571 error.

38572 3. The text region shall be from the current line up to and including the next *count* –1 lines.

38573 Otherwise, the buffer text mode and text region shall be as specified by the motion command.

38574 If in open mode, and the current line is deleted, and the line remains on the display, an '@'
38575 character shall be displayed as the first glyph of that line.

38576 Delete the region of text into *buffer*, if specified, and into the unnamed buffer. If the text to be
38577 deleted contains characters from more than a single line, or the buffer text is in line mode, the
38578 deleted text shall be copied into the numeric buffers, as well.

38579 *Current line*: Set to the first text region line that appears in the edit buffer, unless that line has
38580 been deleted, in which case it shall be set to the last line in the edit buffer, or line 1 if the edit
38581 buffer is empty.

38582 *Current column*:

38583 1. If the line is empty, set to column position 1.

38584 2. Otherwise, if the buffer text is in line mode or the motion was from the cursor toward the
38585 end of the edit buffer:

38586 a. If a character from the current line is displayed in the current column, set to the last
38587 column that displays any portion of that character.

38588 b. Otherwise, set to the last column in which any portion of any character in the line is
38589 displayed.

38590 3. Otherwise, if a character is displayed in the column that began the text region, set to the
38591 last column that displays any portion of that character.

38592 4. Otherwise, set to the last column in which any portion of any character in the line is
38593 displayed.

38594 **Delete to End-of-Line**

38595 *Synopsis*: **[***buffer***]** D

38596 Delete the text from the current position to the end of the current line; equivalent to the *vi*
38597 command:

38598 **[***buffer***]** d$

38599 **Move to End-of-Word**

38600 *Synopsis*: **[***count***]** e

38601 With the exception that words are used instead of bigwords as the delimiter, this command shall
38602 be equivalent to the **E** command.

38603 **Move to End-of-Bigword**

38604 *Synopsis*: **[***count***]** E

38605 If the edit buffer is empty it shall be an error. If less than *count* bigwords end between the cursor
38606 and the end of the edit buffer, *count* shall be adjusted to the number of bigword endings between
38607 the cursor and the end of the edit buffer.

38608 If used as a motion command:

38609 1. The text region shall be from the last character of the *count*th next bigword up to and
38610 including the cursor character.

38611 2. Any text copied to a buffer shall be in character mode.

38612 If not used as a motion command:

38613 *Current line*: Set to the line containing the current column.

38614 *Current column*: Set to the last column upon which any part of the last character of the *count*th
38615 next bigword is displayed.

38616 **Find Character in Current Line (Forward)**

38617 *Synopsis*: **[***count***]** f *character*

38618 It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

38619 If used as a motion command:

38620 1. The text range shall be from the cursor character up to and including the *count*th
38621 occurrence of the specified character after the cursor.

38622 2. Any text copied to a buffer shall be in character mode.

38623 If not used as a motion command:

38624 *Current line*: Unchanged.

38625 *Current column*: Set to the last column in which any portion of the *count*th occurrence of the
38626 specified character after the cursor appears in the line.

38627 **Find Character in Current Line (Reverse)**

38628 *Synopsis*: **[***count***]** F *character*

38629 It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

38630 If used as a motion command:

38631 1. The text region shall be from the *count*th occurrence of the specified character before the
38632 cursor, up to, but not including the cursor character.

38633 2. Any text copied to a buffer shall be in character mode.

38634 If not used as a motion command:

38635 *Current line*: Unchanged.

38636 *Current column*: Set to the last column in which any portion of the *count*th occurrence of the
38637 specified character before the cursor appears in the line.

38638 **Move to Line**

38639 *Synopsis*: **[***count***] G**

38640 If *count* is not specified, it shall default to the last line of the edit buffer. If *count* is greater than
38641 the last line of the edit buffer, it shall be an error.

38642 If used as a motion command:

38643 1. The text region shall be from the cursor line up to and including the specified line.

38644 2. Any text copied to a buffer shall be in line mode.

38645 If not used as a motion command:

38646 *Current line*: Set to *count* if *count* is specified; otherwise, the last line.

38647 *Current column*: Set to non-<blank>.

38648 **Move to Top of Screen**

38649 *Synopsis*: **[***count***] H**

38650 If the beginning of the line *count* greater than the first line of which any portion appears on the
38651 display does not exist, it shall be an error.

38652 If used as a motion command:

38653 1. If in open mode, the text region shall be the current line.

38654 2. Otherwise, the text region shall be from the starting line up to and including (the first line
38655 of the display + *count* –1).

38656 3. Any text copied to a buffer shall be in line mode.

38657 If not used as a motion command:

38658 If in open mode, this command shall set the current column to non-<blank> and do nothing else.

38659 Otherwise, it shall set the current line and current column as follows.

38660 *Current line*: Set to (the first line of the display + *count* –1).

38661 *Current column*: Set to non-<blank>.

38662 **Insert Before Cursor**

38663 *Synopsis*: **[***count***] i**

38664 Enter text input mode before the current cursor position. No characters already in the edit buffer
38665 shall be affected by this command. A *count* shall cause the input text to be appended *count* –1
38666 more times to the end of the input.

38667 *Current line/column*: As specified for the text input commands (see **Input Mode Commands in vi**
38668 (on page 3220)).

38669 **Insert at Beginning of Line**

38670 *Synopsis*: **[***count***]** I

38671 This command shall be equivalent to the *vi* command ^[*count*]**i** command.

38672 **Join**

38673 *Synopsis*: **[***count***]** J

38674 If the current line is the last line in the edit buffer, it shall be an error.

38675 This command shall be equivalent to the *ex* **join** command with no addresses, and an *ex*
38676 command *count* value of 1 if *count* was not specified or if a *count* of 1 was specified, and an *ex*
38677 command *count* value of *count* –1 for any other value of *count*, except that the current line and
38678 column shall be set as follows.

38679 *Current line*: Unchanged.

38680 *Current column*: The last column in which any portion of the character following the last
38681 character in the initial line is displayed, or the last non-<newline> in the line if no characters
38682 were appended.

38683 **Move to Bottom of Screen**

38684 *Synopsis*: **[***count***]** L

38685 If the beginning of the line count less than the last line of which any portion appears on the
38686 display does not exist, it shall be an error.

38687 If used as a motion command:

38688 1. If in open mode, the text region shall be the current line.

38689 2. Otherwise, the text region shall include all lines from the starting cursor line to (the last
38690 line of the display –(*count* –1)).

38691 3. Any text copied to a buffer shall be in line mode.

38692 If not used as a motion command:

38693 1. If in open mode, this command shall set the current column to non-<blank> and do
38694 nothing else.

38695 2. Otherwise, it shall set the current line and current column as follows.

38696 *Current line*: Set to (the last line of the display –(*count* –1)).

38697 *Current column*: Set to non-<blank>.

38698 **Mark Position**

38699 *Synopsis*: m *letter*

38700 This command shall be equivalent to the *ex* **mark** command with the specified character as an
38701 argument.

38702 **Move to Middle of Screen**

38703 *Synopsis*: `M`

38704 The middle line of the display shall be calculated as follows:

38705 `(the top line of the display) + (((number of lines displayed) +1) /2) –1`

38706 If used as a motion command:

38707 1. If in open mode, the text region shall be the current line.

38708 2. Otherwise, the text region shall include all lines from the starting cursor line up to and
38709 including the middle line of the display.

38710 3. Any text copied to a buffer shall be in line mode.

38711 If not used as a motion command:

38712 If in open mode, this command shall set the current column to non-<blank> and do nothing else.

38713 Otherwise, it shall set the current line and current column as follows.

38714 *Current line*: Set to the middle line of the display.

38715 *Current column*: Set to non-<blank>.

38716 **Repeat Regular Expression Find (Forward)**

38717 *Synopsis*: `n`

38718 If the remembered search direction was forward, the **n** command shall be equivalent to the *vi* **/**
38719 command with no characters entered by the user. Otherwise, it shall be equivalent to the *vi* **?**
38720 command with no characters entered by the user.

38721 If the **n** command is used as a motion command for the **!** command, the editor shall not enter
38722 text input mode on the last line on the screen, and shall behave as if the user entered a single `'!'`
38723 character as the text input.

38724 **Repeat Regular Expression Find (Reverse)**

38725 *Synopsis*: `N`

38726 Scan for the next match of the last pattern given to **/** or **?**, but in the reverse direction; this is the
38727 reverse of **n**.

38728 If the remembered search direction was forward, the **N** command shall be equivalent to the *vi* **?**
38729 command with no characters entered by the user. Otherwise, it shall be equivalent to the *vi* **/**
38730 command with no characters entered by the user. If the **N** command is used as a motion
38731 command for the **!** command, the editor shall not enter text input mode on the last line on the
38732 screen, and shall behave as if the user entered a single **!** character as the text input.

38733 **Insert Empty Line Below**

38734 *Synopsis*: `o`

38735 Enter text input mode in a new line appended after the current line. A *count* shall cause the input
38736 text to be appended *count* –1 more times to the end of the already added text, each time starting
38737 on a new, appended line.

38738 *Current line/column*: As specified for the text input commands (see **Input Mode Commands in vi**
38739 (on page 3220)).

38740 **Insert Empty Line Above**

38741 *Synopsis*: `O`

38742 Enter text input mode in a new line inserted before the current line. A *count* shall cause the input
38743 text to be appended *count* –1 more times to the end of the already added text, each time starting
38744 on a new, appended line.

38745 *Current line/column*: As specified for the text input commands (see **Input Mode Commands in vi**
38746 (on page 3220)).

38747 **Put from Buffer Following**

38748 *Synopsis*: `[`*`buffer`*`]` `p`

38749 If no *buffer* is specified, the unnamed buffer shall be used.

38750 If the buffer text is in line mode, the text shall be appended below the current line, and each line
38751 of the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be
38752 appended *count* –1 more times to the end of the already added text, each time starting on a new,
38753 appended line.

38754 If the buffer text is in character mode, the text shall be appended into the current line after the
38755 cursor, and each line of the buffer other than the first and last shall become a new line in the edit
38756 buffer. A *count* shall cause the buffer text to be appended *count* –1 more times to the end of the
38757 already added text, each time starting after the last added character.

38758 *Current line*: If the buffer text is in line mode, set the line to line +1; otherwise, unchanged.

38759 *Current column*: If the buffer text is in line mode:

38760 1. If there is a non-<blank> in the first line of the buffer, set to the last column on which any
38761 portion of the first non-<blank> in the line is displayed.

38762 2. If there is no non-<blank> in the first line of the buffer, set to the last column on which any
38763 portion of the last non-<newline> in the first line of the buffer is displayed.

38764 If the buffer text is in character mode:

38765 1. If the text in the buffer is from more than a single line, then set to the last column on which
38766 any portion of the first character from the buffer is displayed.

38767 2. Otherwise, if the buffer is the unnamed buffer, set to the last column on which any portion
38768 of the last character from the buffer is displayed.

38769 3. Otherwise, set to the first column on which any portion of the first character from the
38770 buffer is displayed.

38771 **Put from Buffer Before**

38772 *Synopsis*: `[`*`buffer`*`]` `P`

38773 If no *buffer* is specified, the unnamed buffer shall be used.

38774 If the buffer text is in line mode, the text shall be inserted above the current line, and each line of
38775 the buffer shall become a new line in the edit buffer. A *count* shall cause the buffer text to be
38776 appended *count* –1 more times to the end of the already added text, each time starting on a new,
38777 appended line.

38778 If the buffer text is in character mode, the text shall be inserted into the current line before the
38779 cursor, and each line of the buffer other than the first and last shall become a new line in the edit
38780 buffer. A *count* shall cause the buffer text to be appended *count* –1 more times to the end of the

already added text, each time starting after the last added character.

Current line: Unchanged.

Current column: If the buffer text is in line mode:

1. If there is a non-<blank> in the first line of the buffer, set to the last column on which any portion of that character is displayed.
2. If there is no non-<blank> in the first line of the buffer, set to the last column on which any portion of the last non-<newline> in the first line of the buffer is displayed.

If the buffer text is in character mode:

1. If the buffer is the unnamed buffer, set to the last column on which any portion of the last character from the buffer is displayed.
2. Otherwise, set to the first column on which any portion of the first character from the buffer is displayed.

Enter ex Mode

Synopsis: `Q`

Leave visual or open mode and enter *ex* command mode.

Current line: Unchanged.

Current column: Unchanged.

Replace Character

Synopsis: `[`*count*`] r` *character*

Replace the *count* characters at and after the cursor with the specified character. If there are less than *count* non-<newline>s at and after the cursor on the line, it shall be an error.

If character is <control>-V, any next character other than the <newline> shall be stripped of any special meaning and used as a literal character.

If character is <ESC>, no replacement shall be made and the current line and current column shall be unchanged.

If character is <carriage-return> or <newline>, *count* new lines shall be appended to the current line. All but the last of these lines shall be empty. *count* characters at and after the cursor shall be discarded, and any remaining characters after the cursor in the current line shall be moved to the last of the new lines. If the **autoindent** edit option is set, they shall be preceded by the same number of **autoindent** characters found on the line from which the command was executed.

Current line: Unchanged unless the replacement character is a <carriage-return> or <newline>, in which case it shall be set to line + *count*.

Current column: Set to the last column position on which a portion of the last replaced character is displayed, or if the replacement character caused new lines to be created, set to non-<blank>.

38815 **Replace Characters**

38816 *Synopsis*: `R`

38817 Enter text input mode at the current cursor position possibly replacing text on the current line. A |
38818 *count* shall cause the input text to be appended *count* –1 more times to the end of the input.

38819 *Current line/column*: As specified for the text input commands (see **Input Mode Commands in vi**
38820 (on page 3220)).

38821 **Substitute Character**

38822 *Synopsis*: **[***buffer***][***count***]** `s`

38823 This command shall be equivalent to the *vi* command:

38824 **[***buffer***][***count***]** `c<space>`

38825 **Substitute Lines**

38826 *Synopsis*: **[***buffer***][***count***]** `S`

38827 This command shall be equivalent to the *vi* command:

38828 **[***buffer***][***count***]** `c_`

38829 **Move Cursor to Before Character (Forward)**

38830 *Synopsis*: **[***count***]** `t` *character*

38831 It shall be an error if *count* occurrences of the character do not occur after the cursor in the line.

38832 If used as a motion command:

38833 1. The text region shall be from the cursor up to but not including the *count*th occurrence of
38834 the specified character after the cursor.

38835 2. Any text copied to a buffer shall be in character mode.

38836 If not used as a motion command:

38837 *Current line*: Unchanged.

38838 *Current column*: Set to the last column in which any portion of the character before the *count*th
38839 occurrence of the specified character after the cursor appears in the line.

38840 **Move Cursor to After Character (Reverse)**

38841 *Synopsis*: **[***count***]** `T` *character*

38842 It shall be an error if *count* occurrences of the character do not occur before the cursor in the line.

38843 If used as a motion command:

38844 1. If the character before the cursor is the specified character, it shall be an error.

38845 2. The text region shall be from the character before the cursor up to but not including the
38846 *count*th occurrence of the specified character before the cursor.

38847 3. Any text copied to a buffer shall be in character mode.

38848 If not used as a motion command:

Current line: Unchanged.

Current column: Set to the last column in which any portion of the character after the *count*th occurrence of the specified character before the cursor appears in the line.

Undo

Synopsis: `u`

This command shall be equivalent to the *ex* **undo** command except that the current line and current column shall be set as follows:

Current line: Set to the first line added or changed if any; otherwise, move to the line preceding any deleted text if one exists; otherwise, move to line 1.

Current column: If undoing an *ex* command, set to the first non-<blank>.

Otherwise, if undoing a text input command:

1. If the command was an **C**, **c**, **O**, **o**, **R**, **S**, or **s** command, the current column shall be set to the value it held when the text input command was entered.
2. Otherwise, set to the last column in which any portion of the first character after the deleted text is displayed, or, if no non-<newline>s follow the text deleted from this line, set to the last column in which any portion of the last non-<newline> in the line is displayed, or 1 if the line is empty.

Otherwise, if a single line was modified (that is, not added or deleted) by the **u** command:

1. If text was added or changed, set to the last column in which any portion of the first character added or changed is displayed.
2. If text was deleted, set to the last column in which any portion of the first character after the deleted text is displayed, or, if no non-<newline>s follow the deleted text, set to the last column in which any portion of the last non-<newline> in the line is displayed, or 1 if the line is empty.

Otherwise, set to non-<blank>.

Undo Current Line

Synopsis: `U`

Restore the current line to its state immediately before the most recent time that it became the current line.

Current line: Unchanged.

Current column: Set to the first column in the line in which any portion of the first character in the line is displayed.

Move to Beginning of Word

Synopsis: `[`*count*`] w`

With the exception that words are used as the delimiter instead of bigwords, this command shall be equivalent to the **W** command.

38885 **Move to Beginning of Bigword**

38886 *Synopsis*: `[count] W`

38887 If the edit buffer is empty, it shall be an error. If there are less than *count* bigwords between the
38888 cursor and the end of the edit buffer, *count* shall be adjusted to move the cursor to the last
38889 bigword in the edit buffer.

38890 If used as a motion command:

38891 1. If the associated command is **c**, *count* is 1, and the cursor is on a <blank>, the region of text
38892 shall be the current character and no further action shall be taken.

38893 2. If there are less than *count* bigwords between the cursor and the end of the edit buffer, then
38894 the command shall succeed, and the region of text shall include the last character of the
38895 edit buffer.

38896 3. If there are <blank>s or an end-of-line that precede the *count*th bigword, and the associated
38897 command is **c**, the region of text shall be up to and including the last character before the
38898 preceding <blank>s or end-of-line.

38899 4. If there are <blank>s or an end-of-line that precede the bigword, and the associated
38900 command is **d** or **y**, the region of text shall be up to and including the last <blank> the start
38901 of the bigword or end-of-line.

38902 5. Any text copied to a buffer shall be in character mode.

38903 If not used as a motion command:

38904 1. If the cursor is on the last character of the edit buffer, it shall be an error.

38905 *Current line*: Set to the line containing the current column.

38906 *Current column*: Set to the last column in which any part of the first character of the *count*th next
38907 bigword is displayed.

38908 **Delete Character at Cursor**

38909 *Synopsis*: `[buffer][count] x`

38910 Delete the *count* characters at and after the current character into *buffer*, if specified, and into the
38911 unnamed buffer.

38912 If the line is empty, it shall be an error. If there are less than *count* non-<newline>s at and after
38913 the cursor on the current line, *count* shall be adjusted to the number of non-<newline>s at and
38914 after the cursor.

38915 *Current line*: Unchanged.

38916 *Current column*: If the line is empty, set to column position 1. Otherwise, if there were *count* or
38917 less non-<newline>s at and after the cursor on the current line, set to the last column that
38918 displays any part of the last non-<newline> of the line. Otherwise, unchanged.

38919 **Delete Character Before Cursor**

38920 *Synopsis*: **[***buffer***][***count***]** X

38921 Delete the *count* characters before the current character into *buffer*, if specified, and into the
38922 unnamed buffer.

38923 If there are no characters before the current character on the current line, it shall be an error. If
38924 there are less than *count* previous characters on the current line, *count* shall be adjusted to the
38925 number of previous characters on the line.

38926 *Current line*: Unchanged.

38927 *Current column*: Set to (*current column – the width of the deleted characters*).

38928 **Yank**

38929 *Synopsis*: **[***buffer***][***count***]** y *motion*

38930 Copy (yank) the region of text into *buffer*, if specified, and into the unnamed buffer.

38931 If the motion command is the **y** command repeated:

38932 1. The buffer shall be in line mode.

38933 2. If there are less than *count* –1 lines after the current line in the edit buffer, it shall be an
38934 error.

38935 3. The text region shall be from the current line up to and including the next *count* –1 lines.

38936 Otherwise, the buffer text mode and text region shall be as specified by the motion command.

38937 *Current line*: If the motion was from the current cursor position toward the end of the edit
38938 buffer, unchanged. Otherwise, set to the first line in the edit buffer that is part of the text region
38939 specified by the motion command.

38940 *Current column*:

38941 1. If the motion was from the current cursor position toward the end of the edit buffer,
38942 unchanged.

38943 2. Otherwise, if the current line is empty, set to column position 1.

38944 3. Otherwise, set to the last column that displays any part of the first character in the file that
38945 is part of the text region specified by the motion command.

38946 **Yank Current Line**

38947 *Synopsis*: **[***buffer***][***count***]** Y

38948 This command shall be equivalent to the *vi* command:

38949 **[***buffer***][***count***]** y_

38950 **Redraw Window**

38951 If in open mode, the **z** command shall have the Synopsis:

38952 *Synopsis*: **[***count***]** z

38953 If *count* is not specified, it shall default to the **window** edit option –1. The **z** command shall be
38954 equivalent to the *ex* **z** command, with a type character of **=** and a *count* of *count* –2, except that
38955 the current line and current column shall be set as follows, and the **window** edit option shall not
38956 be affected. If the calculation for the *count* argument would result in a negative number, the
38957 *count* argument to the *ex* **z** command shall be zero. A blank line shall be written after the last line
38958 is written.

38959 *Current line*: Unchanged.

38960 *Current column*: Unchanged.

38961 If not in open mode, the **z** command shall have the following Synopsis:

38962 *Synopsis*: **[***line***]** z **[***count***]** *character*

38963 If *line* is not specified, it shall default to the current line. If *line* is specified, but is greater than the
38964 number of lines in the edit buffer, it shall default to the number of lines in the edit buffer.

38965 If *count* is specified, the value of the **window** edit option shall be set to *count* (as described in the
38966 *ex* **window** command), and the screen shall be redrawn.

38967 *line* shall be placed as specified by the following characters:

38968 <newline>, <carriage-return>
38969 Place the beginning of the line on the first line of the display.

38970 . Place the beginning of the line in the center of the display. The middle line of the display
38971 shall be calculated as described for the **M** command.

38972 – Place an unspecified portion of the line on the last line of the display.

38973 + If *line* was specified, equivalent to the <newline> case. If *line* was not specified, display a
38974 screen where the first line of the display shall be (current last line) +1. If there are no lines
38975 after the last line in the display, it shall be an error.

38976 ^ If *line* was specified, display a screen where the last line of the display shall contain an
38977 unspecified portion of the first line of a display that had an unspecified portion of the
38978 specified line on the last line of the display. If this calculation results in a line before the
38979 beginning of the edit buffer, display the first screen of the edit buffer.

38980 Otherwise, display a screen where the last line of the display shall contain an unspecified
38981 portion of (current first line –1). If this calculation results in a line before the beginning of
38982 the edit buffer, it shall be an error.

38983 *Current line*: If *line* and the `'^'` character were specified:

38984 1. If the first screen was displayed as a result of the command attempting to display lines
38985 before the beginning of the edit buffer: if the first screen was already displayed,
38986 unchanged; otherwise, set to (current first line –1).

38987 2. Otherwise, set to the last line of the display.

38988 If *line* and the `'+'` character were specified, set to the first line of the display.

38989 Otherwise, if *line* was specified, set to *line*.

Otherwise, unchanged.

Current column: Set to non-<blank>.

Exit

Synopsis: ZZ

This command shall be equivalent to the *ex* **xit** command with no addresses, trailing **!**, or filename (see the *ex* **xit** command).

Input Mode Commands in vi

In text input mode, the current line shall consist of zero or more of the following categories, plus the terminating <newline>:

1. Characters preceding the text input entry point

   Characters in this category shall not be modified during text input mode.

2. **autoindent** characters

   autoindent characters shall be automatically inserted into each line that is created in text input mode, either as a result of entering a <newline> or <carriage-return> while in text input mode, or as an effect of the command itself; for example, **O** or **o** (see the *ex* **autoindent** command), as if entered by the user.

   It shall be possible to erase **autoindent** characters with the <control>-D command; it is unspecified whether they can be erased by <control>-H, <control>-U, and <control>-W characters. Erasing any **autoindent** character turns the glyph into erase-columns and deletes the character from the edit buffer, but does not change its representation on the screen.

3. Text input characters

   Text input characters are the characters entered by the user. Erasing any text input character turns the glyph into erase-columns and deletes the character from the edit buffer, but does not change its representation on the screen.

   Each text input character entered by the user (that does not have a special meaning) shall be treated as follows:

   a. The text input character shall be appended to the last character in the edit buffer from the first, second, or third categories.

   b. If there are no erase-columns on the screen, the text input command was the **R** command, and characters in the fifth category from the original line follow the cursor, the next such character shall be deleted from the edit buffer. If the **slowopen** edit option is not set, the corresponding glyph on the screen shall become erase-columns.

   c. If there are erase-columns on the screen, as many columns as they occupy, or as are necessary, shall be overwritten to display the text input character. (If only part of a multi-column glyph is overwritten, the remainder shall be left on the screen, and continue to be treated as erase-columns; it is unspecified whether the remainder of the glyph is modified in any way.)

   d. If additional display line columns are needed to display the text input character:

      1. If the **slowopen** edit option is set, the text input characters shall be displayed on subsequent display line columns, overwriting any characters displayed in

39032 those columns.

39033 2. Otherwise, any characters currently displayed on or after the column on the
39034 display line where the text input character is to be displayed shall be pushed
39035 ahead the number of display line columns necessary to display the rest of the
39036 text input character.

39037 4. Erase-columns

39038 Erase-columns are not logically part of the edit buffer, appearing only on the screen, and
39039 may be overwritten on the screen by subsequent text input characters. When text input
39040 mode ends, all erase-columns shall no longer appear on the screen.

39041 Erase-columns are initially the region of text specified by the **c** command (see **Change** (on
39042 page 3207)) however, erasing **autoindent** or text input characters causes the glyphs of the
39043 erased characters to be treated as erase-columns.

39044 5. Characters following the text region for the **c** command, or the text input entry point for all
39045 other commands

39046 Characters in this category shall not be modified during text input mode, except as
39047 specified in category 3.b. for the **R** text input command, or as <blank>s deleted when a
39048 <newline> or <carriage-return> is entered.

39049 It is unspecified whether it is an error to attempt to erase past the beginning of a line that was
39050 created by the entry of a <newline> or <carriage-return> during text input mode. If it is not an
39051 error, the editor shall behave as if the erasing character was entered immediately after the last
39052 text input character entered on the previous line, and all of the non-<newline>s on the current
39053 line shall be treated as erase-columns.

39054 When text input mode is entered, or after a text input mode character is entered (except as
39055 specified for the special characters below), the cursor shall be positioned as follows:

39056 1. On the first column that displays any part of the first erase-column, if one exists

39057 2. Otherwise, if the **slowopen** edit option is set, on the first display line column after the last
39058 character in the first, second, or third categories, if one exists

39059 3. Otherwise, the first column that displays any part of the first character in the fifth category,
39060 if one exists

39061 4. Otherwise, the display line column after the last character in the first, second, or third
39062 categories, if one exists

39063 5. Otherwise, on column position 1

39064 The characters that are updated on the screen during text input mode are unspecified, other than
39065 that the last text input character shall always be updated, and, if the **slowopen** edit option is not
39066 set, the current cursor character shall always be updated.

39067 The following specifications are for command characters entered during text input mode.

39068 **NUL**

39069 *Synopsis*: `NUL`

39070 If the first character of the text input is a NUL, the most recently input text shall be input as if
39071 entered by the user, and then text input mode shall be exited. The text shall be input literally;
39072 that is, characters are neither macro or abbreviation expanded, nor are any characters interpreted
39073 in any special manner. It is unspecified whether implementations shall support more than 256
39074 bytes of remembered input text.

39075 **<control>-D**

39076 *Synopsis*: `<control>-D`

39077 The <control>-D character shall have no special meaning when in text input mode for a line-
39078 oriented command (see **Command Descriptions in vi** (on page 3186)).

39079 This command need not be supported on block-mode terminals.

39080 If the cursor does not follow an **autoindent** character, or an **autoindent** character and a `'0'` or
39081 `'^'` character:

39082 1. If the cursor is in column position 1, the <control>-D character shall be discarded and no
39083 further action taken.

39084 2. Otherwise, the <control>-D character shall have no special meaning.

39085 If the last input character was a `'0'`, the cursor shall be moved to column position 1.

39086 Otherwise, if the last input character was a `'^'`, the cursor shall be moved to column position 1.
39087 In addition, the **autoindent** level for the next input line shall be derived from the same line from
39088 which the **autoindent** level for the current input line was derived.

39089 Otherwise, the cursor shall be moved back to the column after the previous shiftwidth (see the
39090 *ex* **shiftwidth** command) boundary.

39091 All of the glyphs on columns between the starting cursor position and (inclusively) the ending
39092 cursor position shall become erase-columns as described in **Input Mode Commands in vi** (on
39093 page 3220).

39094 *Current line*: Unchanged.

39095 *Current column*: Set to 1 if the <control>-D was preceded by a `'^'` or `'0'`; otherwise, set to
39096 (column –1) –((column –2) % **shiftwidth**).

39097 **<control>-H**

39098 *Synopsis*: `<control>-H`

39099 If in text input mode for a line-oriented command, and there are no characters to erase, text
39100 input mode shall be terminated, no further action shall be done for this command, and the
39101 current line and column shall be unchanged.

39102 If there are characters other than **autoindent** characters that have been input on the current line
39103 before the cursor, the cursor shall move back one character.

39104 Otherwise, if there are **autoindent** characters on the current line before the cursor, it is
39105 implementation-defined whether the <control>-H command is an error or if the cursor moves
39106 back one **autoindent** character.

39107 Otherwise, if the cursor is in column position 1 and there are previous lines that have been input,
39108 it is implementation-defined whether the <control>-H command is an error or if it is equivalent

to entering <control>-H after the last input character on the previous input line.

Otherwise, it shall be an error.

All of the glyphs on columns between the starting cursor position and (inclusively) the ending cursor position shall become erase-columns as described in **Input Mode Commands in vi** (on page 3220).

The current erase character (see *stty*) shall cause an equivalent action to the <control>-H command, unless the previously inserted character was a backslash, in which case it shall be as if the literal current erase character had been inserted instead of the backslash.

Current line: Unchanged, unless previously input lines are erased, in which case it shall be set to line –1.

Current column: Set to the first column that displays any portion of the character backed up over.

<newline>

Synopsis:
```
<newline>
<carriage-return>
<control>-J
<control>-M
```

If input was part of a line-oriented command, text input mode shall be terminated and the command shall continue execution with the input provided.

Otherwise, terminate the current line. If there are no characters other than **autoindent** characters on the line, all characters on the line shall be discarded. Otherwise, it is unspecified whether the **autoindent** characters in the line are modified by entering these characters.

Continue text input mode on a new line appended after the current line. If the **slowopen** edit option is set, the lines on the screen below the current line shall not be pushed down, but the first of them shall be cleared and shall appear to be overwritten. Otherwise, the lines of the screen below the current line shall be pushed down.

If the **autoindent** edit option is set, an appropriate number of **autoindent** characters shall be added as a prefix to the line as described by the *ex* **autoindent** edit option.

All columns after the cursor that are erase-columns (as described in **Input Mode Commands in vi** (on page 3220)) shall be discarded.

If the **autoindent** edit option is set, all <blank>s immediately following the cursor shall be discarded.

All remaining characters after the cursor shall be transferred to the new line, positioned after any **autoindent** characters.

Current line: Set to current line +1.

Current column: Set to the first column that displays any portion of the first character after the **autoindent** characters on the new line, if any, or the first column position after the last **autoindent** character, if any, or column position 1.

<control>-T

Synopsis: `<control>-T`

The <control>-T character shall have no special meaning when in text input mode for a line-oriented command (see **Command Descriptions in vi** (on page 3186)).

This command need not be supported on block-mode terminals.

Behave as if the user entered the minimum number of <blank>s necessary to move the cursor forward to the column position after the next **shiftwidth** (see the *ex* **shiftwidth** command) boundary.

Current line: Unchanged.

Current column: Set to *column* + **shiftwidth** – ((column –1) % **shiftwidth**).

<control>-U

Synopsis: `<control>-U`

If there are characters other than **autoindent** characters that have been input on the current line before the cursor, the cursor shall move to the first character input after the **autoindent** characters.

Otherwise, if there are **autoindent** characters on the current line before the cursor, it is implementation-defined whether the <control>-U command is an error or if the cursor moves to the first column position on the line.

Otherwise, if the cursor is in column position 1 and there are previous lines that have been input, it is implementation-defined whether the <control>-U command is an error or if it is equivalent to entering <control>-U after the last input character on the previous input line.

Otherwise, it shall be an error.

All of the glyphs on columns between the starting cursor position and (inclusively) the ending cursor position shall become erase-columns as described in **Input Mode Commands in vi** (on page 3220).

The current *kill* character (see *stty*) shall cause an equivalent action to the <control>-U command, unless the previously inserted character was a backslash, in which case it shall be as if the literal current *kill* character had been inserted instead of the backslash.

Current line: Unchanged, unless previously input lines are erased, in which case it shall be set to line –1.

Current column: Set to the first column that displays any portion of the last character backed up over.

<control>-V

Synopsis: `<control>-V`
`<control>-Q`

Allow the entry of any subsequent character, other than <control>-J or the <newline>, as a literal character, removing any special meaning that it may have to the editor in text input mode. If a <control>-V or <control>-Q is entered before a <control>-J or <newline>, the <control>-V or <control>-Q character shall be discarded, and the <control>-J or <newline> shall behave as described in the <newline> command character during input mode.

For purposes of the display only, the editor shall behave as if a '^' character was entered, and the cursor shall be positioned as if overwriting the '^' character. When a subsequent character is entered, the editor shall behave as if that character was entered instead of the original <control>-V or <control>-Q character.

Current line: Unchanged.

Current column: Unchanged.

<control>-W

Synopsis: `<control>-W`

If there are characters other than **autoindent** characters that have been input on the current line before the cursor, the cursor shall move back over the last word preceding the cursor (including any <blank>s between the end of the last word and the current cursor); the cursor shall not move to before the first character after the end of any **autoindent** characters.

Otherwise, if there are **autoindent** characters on the current line before the cursor, it is implementation-defined whether the <control>-W command is an error or if the cursor moves to the first column position on the line.

Otherwise, if the cursor is in column position 1 and there are previous lines that have been input, it is implementation-defined whether the <control>-W command is an error or if it is equivalent to entering <control>-W after the last input character on the previous input line.

Otherwise, it shall be an error.

All of the glyphs on columns between the starting cursor position and (inclusively) the ending cursor position shall become erase-columns as described in **Input Mode Commands in vi** (on page 3220).

Current line: Unchanged, unless previously input lines are erased, in which case it shall be set to line –1.

Current column: Set to the first column that displays any portion of the last character backed up over.

<ESC>

Synopsis: <ESC>

If input was part of a line-oriented command:

1. If *interrupt* was entered, text input mode shall be terminated and the editor shall return to command mode. The terminal shall be alerted.
2. If <ESC> was entered, text input mode shall be terminated and the command shall continue execution with the input provided.

Otherwise, terminate text input mode and return to command mode.

Any **autoindent** characters entered on newly created lines that have no other non-<newline>s shall be deleted.

Any leading **autoindent** and <blank>s on newly created lines shall be rewritten to be the minimum number of <blank>s possible.

The screen shall be redisplayed as necessary to match the contents of the edit buffer.

Current line: Unchanged.

Current column:

1. If there are text input characters on the current line, the column shall be set to the last column where any portion of the last text input character is displayed.
2. Otherwise, if a character is displayed in the current column, unchanged.
3. Otherwise, set to column position 1.

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

When any error is encountered and the standard input is not a terminal device file, *vi* shall not write the file or return to command or text input mode, and shall terminate with a non-zero exit status.

Otherwise, when an unrecoverable error is encountered it shall be equivalent to a SIGHUP asynchronous event.

Otherwise, when an error is encountered, the editor shall behave as specified in **Command Descriptions in vi** (on page 3186).

APPLICATION USAGE

None.

EXAMPLES

None.

RATIONALE

See the RATIONALE for *ex* for more information on *vi*. Major portions of the *vi* utility specification point to *ex* to avoid inadvertent divergence. While *ex* and *vi* have historically been implemented as a single utility, this is not required by IEEE Std 1003.1-200x.

It is recognized that portions of *vi* would be difficult, if not impossible, to implement satisfactorily on a block-mode terminal, or a terminal without any form of cursor addressing, thus it is not a mandatory requirement that such features should work on all terminals. It is the intention, however, that a *vi* implementation should provide the full set of capabilities on all terminals capable of supporting them.

Historically, *vi* exited immediately if the standard input was not a terminal. IEEE Std 1003.1-200x permits, but does not require, this behavior. An end-of-file condition is not equivalent to an end-of-file character. A common end-of-file character, <control>-D, is historically a *vi* command.

The text in the STANDARD OUTPUT section reflects the usage of the verb *display* in this section; some implementations of *vi* use standard output to write to the terminal, but IEEE Std 1003.1-200x does not require that to be the case.

Historically, implementations reverted to open mode if the terminal was incapable of supporting full visual mode. IEEE Std 1003.1-200x requires this behavior. Historically, the open mode of *vi* behaved roughly equivalently to the visual mode, with the exception that only a single line from the edit buffer (one ‘‘buffer line’’) was kept current at any time. This line was normally displayed on the next-to-last line of a terminal with cursor addressing (and the last line performed its normal visual functions for line-oriented commands and messages). In addition, some few commands behaved differently in open mode than in visual mode. IEEE Std 1003.1-200x requires conformance to historical practice.

Historically, *ex* and *vi* implementations have expected text to proceed in the usual European/Latin order of left to right, top to bottom. There is no requirement in IEEE Std 1003.1-200x that this be the case. The specification was deliberately written using words like ''before'', ''after'', ''first'', and ''last'' in order to permit implementations to support the natural text order of the language.

Historically, lines past the end of the edit buffer were marked with single tilde (`'~'`) characters; that is, if the one-based display was 20 lines in length, and the last line of the file was on line one, then lines 2-20 would contain only a single `'~'` character.

Historically, the *vi* editor attempted to display only complete lines at the bottom of the screen (it did display partial lines at the top of the screen). If a line was too long to fit in its entirety at the bottom of the screen, the screen lines where the line would have been displayed were displayed as single `'@'` characters, instead of displaying part of the line. IEEE Std 1003.1-200x permits, but does not require, this behavior. Implementations are encouraged to attempt always to display a complete line at the bottom of the screen when doing scrolling or screen positioning by buffer lines.

Historically, lines marked with `'@'` were also used to minimize output to dumb terminals over slow lines; that is, changes local to the cursor were updated, but changes to lines on the screen that were not close to the cursor were simply marked with an `'@'` sign instead of being updated to match the current text. IEEE Std 1003.1-200x permits, but does not require this feature because it is used ever less frequently as terminals become smarter and connections are faster.

Initialization in ex and vi

Historically, *vi* always had a line in the edit buffer, even if the edit buffer was ''empty''. For example:

1. The *ex* command **=** executed from visual mode wrote ''1'' when the buffer was empty.
2. Writes from visual mode of an empty edit buffer wrote files of a single character (a <newline>), while writes from *ex* mode of an empty edit buffer wrote empty files.
3. Put and read commands into an empty edit buffer left an empty line at the top of the edit buffer.

For consistency, IEEE Std 1003.1-200x does not permit any of these behaviors.

Historically, *vi* did not always return the terminal to its original modes; for example, ICRNL was modified if it was not originally set. IEEE Std 1003.1-200x does not permit this behavior.

Command Descriptions in vi

Motion commands are among the most complicated aspects of *vi* to describe. With some exceptions, the text region and buffer type effect of a motion command on a *vi* command are described on a case-by-case basis. The descriptions of text regions in IEEE Std 1003.1-200x are not intended to imply direction; that is, an inclusive region from line *n* to line *n*+5 is identical to a region from line *n*+5 to line *n*. This is of more than academic interest—movements to marks can be in either direction, and, if the **wrapscan** option is set, so can movements to search points. Historically, lines are always stored into buffers in text order; that is, from the start of the edit buffer to the end. IEEE Std 1003.1-200x requires conformance to historical practice.

Historically, command counts were applied to any associated motion, and were multiplicative to any supplied motion count. For example, **2cw** is the same as **c2w**, and **2c3w** is the same as **c6w**. IEEE Std 1003.1-200x requires this behavior. Historically, *vi* commands that used bigwords, words, paragraphs, and sentences as objects treated groups of empty lines, or lines that contained only <blank>s, inconsistently. Some commands treated them as a single entity, while

others treated each line separately. For example, the **w**, **W**, and **B** commands treated groups of empty lines as individual words; that is, the command would move the cursor to each new empty line. The **e** and **E** commands treated groups of empty lines as a single word; that is, the first use would move past the group of lines. The **b** command would just beep at the user, or if done from the start of the line as a motion command, fail in unexpected ways. If the lines contained only (or ended with) <blank>s, the **w** and **W** commands would just beep at the user, the **E** and **e** commands would treat the group as a single word, and the **B** and **b** commands would treat the lines as individual words. For consistency and simplicity of specification, IEEE Std 1003.1-200x requires that all *vi* commands treat groups of empty or blank lines as a single entity, and that movement through lines ending with <blank>s be consistent with other movements.

Historically, *vi* documentation indicated that any number of double quotes were skipped after punctuation marks at sentence boundaries; however, implementations only skipped single quotes. IEEE Std 1003.1-200x requires both to be skipped.

Historically, the first and last characters in the edit buffer were word boundaries. This historical practice is required by IEEE Std 1003.1-200x.

Historically, *vi* attempted to update the minimum number of columns on the screen possible, which could lead to misleading information being displayed. IEEE Std 1003.1-200x makes no requirements other than that the current character being entered is displayed correctly, leaving all other decisions in this area up to the implementation.

Historically, lines were arbitrarily folded between columns of any characters that required multiple column positions on the screen, with the exception of tabs, which terminated at the right-hand margin. IEEE Std 1003.1-200x permits the former and requires the latter. Implementations that do not arbitrarily break lines between columns of characters that occupy multiple column positions should not permit the cursor to rest on a column that does not contain any part of a character.

The historical *vi* had a problem in that all movements were by buffer lines, not by display or screen lines. This is often the right thing to do; for example, single line movements, such as **j** or **k**, should work on buffer lines. Commands like **dj**, or **j.**, where **.** is a change command, only make sense for buffer lines. It is not, however, the right thing to do for screen motion or scrolling commands like <control>-D, <control>-F, and **H**. If the window is fairly small, using buffer lines in these cases can result in completely random motion; for example, **1**<control>-D can result in a completely changed screen, without any overlap. This is clearly not what the user wanted. The problem is even worse in the case of the **H**, **L**, and **M** commands—as they position the cursor at the first non-<blank> of the line, they may all refer to the same location in large lines, and will result in no movement at all.

In addition, if the line is larger than the screen, using buffer lines can make it impossible to display parts of the line—there are not any commands that do not display the beginning of the line in historical *vi*, and if both the beginning and end of the line cannot be on the screen at the same time, the user suffers. Finally, the page and half-page scrolling commands historically moved to the first non-<blank> in the new line. If the line is approximately the same size as the screen, this is inadequate because the cursor before and after a <control>-D command will refer to the same location on the screen.

Implementations of *ex* and *vi* exist that do not have these problems because the relevant commands (<control>-B, <control>-D, <control>-F, <control>-U, <control>-Y, <control>-E, **H**, **L**, and **M)** operate on display (screen) lines, not (edit) buffer lines.

IEEE Std 1003.1-200x does not permit this behavior by default because the standard developers believed that users would find it too confusing. However, historical practice has been relaxed.

For example, *ex* and *vi* historically attempted, albeit sometimes unsuccessfully, to never put part of a line on the last lines of a screen; for example, if a line would not fit in its entirety, no part of the line was displayed, and the screen lines corresponding to the line contained single '@' characters. This behavior is permitted, but not required by IEEE Std 1003.1-200x, so that it is possible for implementations to support long lines in small screens more reasonably without changing the commands to be oriented to the display (instead of oriented to the buffer). IEEE Std 1003.1-200x also permits implementations to refuse to edit any edit buffer containing a line that will not fit on the screen in its entirety.

The display area (for example, the value of the **window** edit option) has historically been ''grown'', or expanded, to display new text when local movements are done in displays where the number of lines displayed is less than the maximum possible. Expansion has historically been the first choice, when the target line is less than the maximum possible expansion value away. Scrolling has historically been the next choice, done when the target line is less than half a display away, and otherwise, the screen was redrawn. There were exceptions, however, in that *ex* commands generally always caused the screen to be redrawn. IEEE Std 1003.1-200x does not specify a standard behavior because there may be external issues, such as connection speed, the number of characters necessary to redraw as opposed to scroll, or terminal capabilities that implementations will have to accommodate.

The current line in IEEE Std 1003.1-200x maps one-to-one to a buffer line in the file. The current column does not. There are two different column values that are described by IEEE Std 1003.1-200x. The first is the current column value as set by many of the *vi* commands. This value is remembered for the lifetime of the editor. The second column value is the actual position on the screen where the cursor rests. The two are not always the same. For example, when the cursor is backed by a multi-column character, the actual cursor position on the screen has historically been the last column of the character in command mode, and the first column of the character in input mode.

Commands that set the current line, but that do not set the current cursor value (for example, **j** and **k**) attempt to get as close as possible to the remembered column position, so that the cursor tends to restrict itself to a vertical column as the user moves around in the edit buffer. IEEE Std 1003.1-200x requires conformance to historical practice, requiring that the display location of the cursor on the display line be adjusted from the current column value as necessary to support this historical behavior.

Historically, only a single line (and for some terminals, a single line minus 1 column) of characters could be entered by the user for the line oriented commands; that is, **:**, **!**, **/**, or **?**. IEEE Std 1003.1-200x permits, but does not require, this limitation.

Historically, ''soft'' errors in *vi* caused the terminal to be alerted, but no error message was displayed. As a general rule, no error message was displayed for errors in command execution in *vi*, when the error resulted from the user attempting an invalid or impossible action, or when a searched-for object was not found. Examples of soft errors included **h** at the left margin, <control>-B or **[[** at the beginning of the file, **2G** at the end of the file, and so on. In addition, errors such as **%**, **]]**, **}**, **)**, **N**, **n**, **f**, **F**, **t**, and **T** failing to find the searched-for object were soft as well. Less consistently, **/** and **?** displayed an error message if the pattern was not found, **/**, **?**, **N**, and **n** displayed an error message if no previous regular expression had been specified, and **;** did not display an error message if no previous **f**, **F**, **t**, or **T** command had occurred. Also, behavior in this area might reasonably be based on a runtime evaluation of the speed of a network connection. Finally, some implementations have provided error messages for soft errors in order to assist naive users, based on the value of a verbose edit option. IEEE Std 1003.1-200x does not list specific errors for which an error message shall be displayed. Implementations should conform to historical practice in the absence of any strong reason to diverge.

39413 **Page Backwards**

39414 The <control>-B and <control>-F commands historically considered it an error to attempt to
39415 page past the beginning or end of the file, whereas the <control>-D and <control>-U commands
39416 simply moved to the beginning or end of the file. For consistency, IEEE Std 1003.1-200x requires
39417 the latter behavior for all four commands. All four commands still consider it an error if the
39418 current line is at the beginning (<control>-B, <control>-U) or end (<control>-F, <control>-D) of
39419 the file. Historically, the <control>-B and <control>-F commands skip two lines in order to
39420 include overlapping lines when a single command is entered. This makes less sense in the
39421 presence of a *count*, as there will be, by definition, no overlapping lines. The actual calculation
39422 used by historical implementations of the *vi* editor for <control>-B was:

39423 `((current first line) – count x (window edit option)) +2`

39424 and for <control>-F was:

39425 `((current first line) + count x (window edit option)) –2`

39426 This calculation does not work well when intermixing commands with and without counts; for
39427 example, **3**<control>-F is not equivalent to entering the <control>-F command three times, and is
39428 not reversible by entering the <control>-B command three times. For consistency with other *vi*
39429 commands that take counts, IEEE Std 1003.1-200x requires a different calculation.

39430 **Scroll Forward**

39431 The 4BSD and System V implementations of *vi* differed on the initial value used by the **scroll**
39432 command. 4BSD used:

39433 `((window edit option) +1) /2`

39434 while System V used the value of the **scroll** edit option. The System V version is specified by
39435 IEEE Std 1003.1-200x because the standard developers believed that it was more intuitive and
39436 permitted the user a method of setting the scroll value initially without also setting the number
39437 of lines that are displayed.

39438 **Scroll Forward by Line**

39439 Historically, the <control>-E and <control>-Y commands considered it an error if the last and
39440 first lines, respectively, were already on the screen. IEEE Std 1003.1-200x requires conformance
39441 to historical practice. Historically, the <control>-E and <control>-Y commands had no effect in
39442 open mode. For simplicity and consistency of specification, IEEE Std 1003.1-200x requires that
39443 they behave as usual, albeit with a single line screen.

39444 **Clear and Redisplay**

39445 The historical <control>-L command refreshed the screen exactly as it was supposed to be
39446 currently displayed, replacing any `'@'` characters for lines that had been deleted but not
39447 updated on the screen with refreshed `'@'` characters. The intent of the <control>-L command is
39448 to refresh when the screen has been accidentally overwritten; for example, by a **write** command
39449 from another user, or modem noise.

39450 **Redraw Screen**

39451 The historical <control>-R command redisplayed only when necessary to update lines that had
39452 been deleted but not updated on the screen and that were flagged with `'@'` characters. There is
39453 no requirement that the screen be in any way refreshed if no lines of this form are currently
39454 displayed. IEEE Std 1003.1-200x permits implementations to extend this command to refresh
39455 lines on the screen flagged with `'@'` characters because they are too long to be displayed in the
39456 current framework; however, the current line and column need not be modified.

39457 **Search for tagstring**

39458 Historically, the first non-<blank> at or after the cursor was the first character, and all
39459 subsequent characters that were word characters, up to the end of the line, were included. For
39460 example, with the cursor on the leading space or on the `'#'` character in the text `"#bar@"`, the
39461 tag was `"#bar"`. On the character `'b'` it was `"bar"`, and on the `'a'`, it was `"ar"`.
39462 IEEE Std 1003.1-200x requires this behavior.

39463 **Replace Text with Results from Shell Command**

39464 Historically, the **<**, **>**, and **!** commands considered most cursor motions other than line-oriented
39465 motions an error; for example, the command **>/foo<CR>** succeeded, while the command **>l**
39466 failed, even though the text region described by the two commands might be identical. For
39467 consistency, all three commands only consider entire lines and not partial lines, and the region is
39468 defined as any line that contains a character that was specified by the motion.

39469 **Move to Matching Character**

39470 Other matching characters have been left implementation-defined in order to allow extensions
39471 such as matching `'<'` and `'>'` for searching HTML, or **#ifdef**, **#else**, and **#endif** for searching C
39472 source.

39473 **Repeat Substitution**

39474 IEEE Std 1003.1-200x requires that any **c** and **g** flags specified to the previous substitute
39475 command be ignored; however, the **r** flag may still apply, if supported by the implementation.

39476 **Return to Previous (Context or Section)**

39477 The **[[**, **]]**, **(**, **)**, **{**, and **}** commands are all affected by ‘‘section boundaries’’, but in some historical
39478 implementations not all of the commands recognize the same section boundaries. This is a bug,
39479 not a feature, and a unique section-boundary algorithm was not described for each command.
39480 One special case that is preserved is that the sentence command moves to the end of the last line
39481 of the edit buffer while the other commands go to the beginning, in order to preserve the
39482 traditional character cut semantics of the sentence command. Historically, *vi* section boundaries
39483 at the beginning and end of the edit buffer were the first non-<blank> on the first and last lines
39484 of the edit buffer if one exists; otherwise, the last character of the first and last lines of the edit
39485 buffer if one exists. To increase consistency with other section locations, this has been simplified
39486 by IEEE Std 1003.1-200x to the first character of the first and last lines of the edit buffer, or the
39487 first and the last lines of the edit buffer if they are empty.

39488 Sentence boundaries were problematic in the historical *vi*. They were not only the boundaries as
39489 defined for the section and paragraph commands, but they were the first non-<blank> that
39490 occurred after those boundaries, as well. Historically, the *vi* section commands were
39491 documented as taking an optional window size as a *count* preceding the command. This was not
39492 implemented in historical versions, so IEEE Std 1003.1-200x requires that the *count* repeat the
39493 command, for consistency with other *vi* commands.

39494 **Repeat**

39495 Historically, mapped commands other than text input commands could not be repeated using
39496 the **period** command. IEEE Std 1003.1-200x requires conformance to historical practice.

39497 The restrictions on the interpretation of special characters (for example, <control>-H) in the
39498 repetition of text input mode commands is intended to match historical practice. For example,
39499 given the input sequence:

39500 `iab<control>-H<control>-H<control>-Hdef<escape>`

39501 the user should be informed of an error when the sequence is first entered, but not during a
39502 command repetition. The character <control>-T is specifically exempted from this restriction.
39503 Historical implementations of *vi* ignored <control>-T characters that were input in the original
39504 command during command repetition. IEEE Std 1003.1-200x prohibits this behavior.

39505 **Find Regular Expression**

39506 Historically, commands did not affect the line searched to or from if the motion command was a
39507 search (**/**, **?**, **N**, **n**) and the final position was the start/end of the line. There were some special
39508 cases and *vi* was not consistent. IEEE Std 1003.1-200x does not permit this behavior, for
39509 consistency. Historical implementations permitted, but were unable to handle searches as
39510 motion commands that wrapped (that is, due to the edit option **wrapscan**) to the original
39511 location. IEEE Std 1003.1-200x requires that this behavior be treated as an error.

39512 Historically, the syntax `"/RE/0"` was used to force the command to cut text in line mode.
39513 IEEE Std 1003.1-200x requires conformance to historical practice.

39514 Historically, in open mode, a **z** specified to a search command redisplayed the current line
39515 instead of displaying the current screen with the current line highlighted. For consistency and
39516 simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior.

39517 Historically, trailing **z** commands were permitted and ignored if entered as part of a search used
39518 as a motion command. For consistency and simplicity of specification, IEEE Std 1003.1-200x does
39519 not permit this behavior.

39520 **Execute an ex Command**

39521 Historically, *vi* implementations restricted the commands that could be entered on the colon
39522 command line (for example, **append** and **change**), and some other commands were known to
39523 cause them to fail catastrophically. For consistency, IEEE Std 1003.1-200x does not permit these
39524 restrictions. When executing an *ex* command by entering **:**, it is not possible to enter a <newline>
39525 as part of the command because it is considered the end of the command. A different approach
39526 is to enter *ex* command mode by using the *vi* **Q** command (and later resuming visual mode with
39527 the *ex* **vi** command). In *ex* command mode, the single-line limitation does not exist. So, for
39528 example, the following is valid:

39529 ```
Q
39530 s/break here/break\
39531 here/
39532 vi
```

39533 IEEE Std 1003.1-200x requires that, if the *ex* command overwrites any part of the screen that
39534 would be erased by a refresh, *vi* pauses for a character from the user. Historically, this character
39535 could be any character; for example, a character input by the user before the message appeared,
39536 or even a mapped character. This is probably a bug, but implementations that have tried to be
39537 more rigorous by requiring that the user enter a specific character, or that the user enter a
39538 character after the message was displayed, have been forced by user indignation back into
```

historical behavior. IEEE Std 1003.1-200x requires conformance to historical practice.

Shift Left (Right)

Refer to the Rationale for the **!** and **/** commands. Historically, the **<** and **>** commands sometimes moved the cursor to the first non-<blank> (for example if the command was repeated or with **_** as the motion command), and sometimes left it unchanged. IEEE Std 1003.1-200x does not permit this inconsistency, requiring instead that the cursor always move to the first non-<blank>. Historically, the **<** and **>** commands did not support buffer arguments, although some implementations allow the specification of an optional buffer. This behavior is neither required nor disallowed by IEEE Std 1003.1-200x.

Execute

Historically, buffers could execute other buffers, and loops, infinite and otherwise, were possible. IEEE Std 1003.1-200x requires conformance to historical practice. The **buffer* syntax of *ex* is not required in *vi*, because it is not historical practice and has been used in some *vi* implementations to support additional scripting languages.

Reverse Case

Historically, the **~** command ignored any associated *count*, and acted only on the characters in the current line. For consistency with other *vi* commands, IEEE Std 1003.1-200x requires that an associated *count* act on the next *count* characters, and that the command move to subsequent lines if warranted by *count*, to make it possible to modify large pieces of text in a reasonably efficient manner. There exist *vi* implementations that optionally require an associated motion command for the **~** command. Implementations supporting this functionality are encouraged to base it on the **tildedop** edit option and handle the text regions and cursor positioning identically to the **yank** command.

Append

Historically, *count*s specified to the **A**, **a**, **I**, and **i** commands repeated the input of the first line *count* times, and did not repeat the subsequent lines of the input text. IEEE Std 1003.1-200x requires that the entire text input be repeated *count* times.

Move Backward to Preceding Word

Historically, *vi* became confused if word commands were used as motion commands in empty files. IEEE Std 1003.1-200x requires that this be an error. Historical implementations of *vi* had a large number of bugs in the word movement commands, and they varied greatly in behavior in the presence of empty lines, ‘‘words’’ made up of a single character, and lines containing only <blank>s. For consistency and simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior.

Change to End-of-Line

Some historical implementations of the **C** command did not behave as described by IEEE Std 1003.1-200x when the **$** key was remapped because they were implemented by pushing the **$** key onto the input queue and reprocessing it. IEEE Std 1003.1-200x does not permit this behavior. Historically, the **C**, **S**, and **s** commands did not copy replaced text into the numeric buffers. For consistency and simplicity of specification, IEEE Std 1003.1-200x requires that they behave like their respective **c** commands in all respects.

39580 **Delete**

39581 Historically, lines in open mode that were deleted were scrolled up, and an **@** glyph written over
39582 the beginning of the line. In the case of terminals that are incapable of the necessary cursor
39583 motions, the editor erased the deleted line from the screen. IEEE Std 1003.1-200x requires
39584 conformance to historical practice; that is, if the terminal cannot display the '@' character, the
39585 line cannot remain on the screen.

39586 **Delete to End-of-Line**

39587 Some historical implementations of the **D** command did not behave as described by
39588 IEEE Std 1003.1-200x when the **$** key was remapped because they were implemented by pushing
39589 the **$** key onto the input queue and reprocessing it. IEEE Std 1003.1-200x does not permit this
39590 behavior.

39591 **Join**

39592 An historical oddity of *vi* is that the commands **J**, **1J**, and **2J** are all equivalent.
39593 IEEE Std 1003.1-200x requires conformance to historical practice. The *vi* **J** command is specified
39594 in terms of the *ex* **join** command with an *ex* command *count* value. The address correction for a
39595 *count* that is past the end of the edit buffer is necessary for historical compatibility for both *ex*
39596 and *vi*.

39597 **Mark Position**

39598 Historical practice is that only lowercase letters, plus '' and '', could be used to mark a
39599 cursor position. IEEE Std 1003.1-200x requires conformance to historical practice, but encourages
39600 implementations to support other characters as marks as well.

39601 **Repeat Regular Expression Find (Forward and Reverse)**

39602 Historically, the **N** and **n** commands could not be used as motion components for the **c**
39603 command. With the exception of the **cN** command, which worked if the search crossed a line
39604 boundary, the text region would be discarded, and the user would not be in text input mode. For
39605 consistency and simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior.

39606 **Insert Empty Line (Below and Above)**

39607 Historically, counts to the **O** and **o** commands were used as the number of physical lines to
39608 open, if the terminal was dumb and the **slowopen** option was not set. This was intended to
39609 minimize traffic over slow connections and repainting for dumb terminals. IEEE Std 1003.1-200x
39610 does not permit this behavior, requiring that a *count* to the open command behave as for other
39611 text input commands. This change to historical practice was made for consistency, and because a
39612 superset of the functionality is provided by the **slowopen** edit option.

39613 **Put from Buffer (Following and Before)**

39614 Historically, *count*s to the **p** and **P** commands were ignored if the buffer was a line mode buffer,
39615 but were (mostly) implemented as described in IEEE Std 1003.1-200x if the buffer was a
39616 character mode buffer. Because implementations exist that do not have this limitation, and
39617 because pasting lines multiple times is generally useful, IEEE Std 1003.1-200x requires that *count*
39618 be supported for all **p** and **P** commands.

39619 Historical implementations of *vi* were widely known to have major problems in the **p** and **P**
39620 commands, particularly when unusual regions of text were copied into the edit buffer. The
39621 standard developers viewed these as bugs, and they are not permitted for consistency and

simplicity of specification.

Historically, a **P** or **p** command (or an *ex* **put** command executed from open or visual mode) executed in an empty file, left an empty line as the first line of the file. For consistency and simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior.

Replace Character

Historically, the **r** command did not correctly handle the *erase* and *word erase* characters as arguments, nor did it handle an associated *count* greater than 1 with a <carriage-return> argument, for which it replaced *count* characters with a single <newline>. IEEE Std 1003.1-200x does not permit these inconsistencies.

Historically, the **r** command permitted the <control>-V escaping of entered characters, such as <ESC> and the <carriage-return>; however, it required two leading <control>-V characters instead of one. IEEE Std 1003.1-200x requires that this be changed for consistency with the other text input commands of *vi*.

Historically, it is an error to enter the **r** command if there are less than *count* characters at or after the cursor in the line. While a reasonable and unambiguous extension would be to permit the **r** command on empty lines, it would require that too large a *count* be adjusted to match the number of characters at or after the cursor for consistency, which is sufficiently different from historical practice to be avoided. IEEE Std 1003.1-200x requires conformance to historical practice.

Replace Characters

Historically, if there were **autoindent** characters in the line on which the **R** command was run, and **autoindent** was set, the first <newline> would be properly indented and no characters would be replaced by the <newline>. Each additional <newline> would replace *n* characters, where *n* was the number of characters that were needed to indent the rest of the line to the proper indentation level. This behavior is a bug and is not permitted by IEEE Std 1003.1-200x.

Undo

Historical practice for cursor positioning after undoing commands was mixed. In most cases, when undoing commands that affected a single line, the cursor was moved to the start of added or changed text, or immediately after deleted text. However, if the user had moved from the line being changed, the column was either set to the first non-<blank>, returned to the origin of the command, or remained unchanged. When undoing commands that affected multiple lines or entire lines, the cursor was moved to the first character in the first line restored. As an example of how inconsistent this was, a search, followed by an **o** text input command, followed by an **undo** would return the cursor to the location where the **o** command was entered, but a **cw** command followed by an **o** command followed by an **undo** would return the cursor to the first non-<blank> of the line. IEEE Std 1003.1-200x requires the most useful of these behaviors, and discards the least useful, in the interest of consistency and simplicity of specification.

39659 **Yank**

39660 Historically, the **yank** command did not move to the end of the motion if the motion was in the
39661 forward direction. It moved to the end of the motion if the motion was in the backward
39662 direction, except for the **_** command, or for the **G** and **'** commands when the end of the motion
39663 was on the current line. This was further complicated by the fact that for a number of motion
39664 commands, the **yank** command moved the cursor but did not update the screen; for example, a
39665 subsequent command would move the cursor from the end of the motion, even though the
39666 cursor on the screen had not reflected the cursor movement for the **yank** command.
39667 IEEE Std 1003.1-200x requires that all **yank** commands associated with backward motions move
39668 the cursor to the end of the motion for consistency, and specifically, to make **'** commands as
39669 motions consistent with search patterns as motions.

39670 **Yank Current Line**

39671 Some historical implementations of the **Y** command did not behave as described by
39672 IEEE Std 1003.1-200x when the `'_'` key was remapped because they were implemented by
39673 pushing the `'_'` key onto the input queue and reprocessing it. IEEE Std 1003.1-200x does not
39674 permit this behavior.

39675 **Redraw Window**

39676 Historically, the **z** command always redrew the screen. This is permitted but not required by
39677 IEEE Std 1003.1-200x, because of the frequent use of the **z** command in macros such as **map n nz.**
39678 for screen positioning, instead of its use to change the screen size. The standard developers
39679 believed that expanding or scrolling the screen offered a better interface for users. The ability to
39680 redraw the screen is preserved if the optional new window size is specified, and in the
39681 <control>-L and <control>-R commands.

39682 The semantics of **z^** are confusing at best. Historical practice is that the screen before the screen
39683 that ended with the specified line is displayed. IEEE Std 1003.1-200x requires conformance to
39684 historical practice.

39685 Historically, the **z** command would not display a partial line at the top or bottom of the screen. If
39686 the partial line would normally have been displayed at the bottom of the screen, the command
39687 worked, but the partial line was replaced with `'@'` characters. If the partial line would normally
39688 have been displayed at the top of the screen, the command would fail. For consistency and
39689 simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior.

39690 Historically, the **z** command with a line specification of 1 ignored the command. For consistency
39691 and simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior.

39692 Historically, the **z** command did not set the cursor column to the first non-<blank> for the
39693 character if the first screen was to be displayed, and was already displayed. For consistency and
39694 simplicity of specification, IEEE Std 1003.1-200x does not permit this behavior.

39695 **Input Mode Commands in vi**

39696 Historical implementations of *vi* did not permit the user to erase more than a single line of input, |
39697 or to use normal erase characters such as *line erase*, *worderase*, and *erase* to erase **autoindent**
39698 characters. As there exist implementations of *vi* that do not have these limitations, both
39699 behaviors are permitted, but only historical practice is required. In the case of these extensions,
39700 *vi* is required to pause at the **autoindent** and previous line boundaries.

39701 Historical implementations of *vi* updated only the portion of the screen where the current cursor
39702 character was displayed. For example, consider the *vi* input keystrokes:

```
iabcd<escape>0C<tab>
```

Historically, the <tab> would overwrite the characters `"abcd"` when it was displayed. Other implementations replace only the `'a'` character with the <tab>, and then push the rest of the characters ahead of the cursor. Both implementations have problems. The historical implementation is probably visually nicer for the above example; however, for the keystrokes:

```
iabcd<ESC>0R<tab><ESC>
```

the historical implementation results in the string `"bcd"` disappearing and then magically reappearing when the <ESC> character is entered. IEEE Std 1003.1-200x requires the former behavior when overwriting erase-columns; that is, overwriting characters that are no longer logically part of the edit buffer, and the latter behavior otherwise.

Historical implementations of *vi* discarded the <control>-D and <control>-T characters when they were entered at places where their command functionality was not appropriate. IEEE Std 1003.1-200x requires that the <control>-T functionality always be available, and that <control>-D be treated as any other key when not operating on **autoindent** characters.

NUL

Some historical implementations of *vi* limited the number of characters entered using the NUL input character to 256 bytes. IEEE Std 1003.1-200x permits this limitation; however, implementations are encouraged to remove this limit.

<control>-D

See also Rationale for the input mode command <newline>. The hidden assumptions in the <control>-D command (and in the *vi* **autoindent** specification in general) is that <space>s take up a single column on the screen and that <tab>s are comprised of an integral number of <space>s.

<newline>

Implementations are permitted to rewrite **autoindent** characters in the line when <newline>, <carriage-return>, <control>-D, and <control>-T are entered, or when the **shift** commands are used, because historical implementations have both done so and found it necessary to do so. For example, a <control>-D when the cursor is preceded by a single <tab>, with **tabstop** set to 8, and **shiftwidth** set to 3, will result in the <tab> being replaced by several <space>s.

<control>-T

See also the Rationale for the input mode command <newline>. Historically, <control>-T only worked if no non-<blank>s had yet been input in the current input line. In addition, the characters inserted by <control>-T were treated as **autoindent** characters, and could not be erased using normal user erase characters. Because implementations exist that do not have these limitations, and as moving to a column boundary is generally useful, IEEE Std 1003.1-200x requires that both limitations be removed.

<control>-V

Historically, *vi* used **^V**, regardless of the value of the literal-next character of the terminal. IEEE Std 1003.1-200x requires conformance to historical practice.

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 to off. (In addition, some historical implementations of *vi* explicitly set **ixon** mode to on, so it was difficult for the user to set it to off.) Any of the command characters described in IEEE Std 1003.1-200x can be made ineffective by their selection as *termios* control characters, using the *stty* utility or other methods described in the System Interfaces volume of IEEE Std 1003.1-200x.

<ESC>

Historically, SIGINT alerted the terminal when used to end input mode. This behavior is permitted, but not required, by IEEE Std 1003.1-200x.

FUTURE DIRECTIONS

None.

SEE ALSO

ex, *stty*

CHANGE HISTORY

First released in Issue 2.

Issue 5

FUTURE DIRECTIONS section added.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The APPLICATION USAGE section is added.

The obsolescent SYNOPSIS is removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **reindent** command description is added.

The *vi* utility has been extensively rewritten for alignment with the IEEE P1003.2b draft standard.

IEEE PASC Interpretations 1003.2 #57, #62, #63, #64, #78, and #188 are applied.

IEEE PASC Interpretation 1003.2 #207 is applied, clarifying the description of the **R** command in a manner similar to the descriptions of other text input mode commands such as **i**, **o**, and **O**.

NAME

wait — await process completion

SYNOPSIS

```
wait [pid...]
```

DESCRIPTION

When an asynchronous list (see Section 2.9.3.1 (on page 2252)) is started by the shell, the process ID of the last command in each element of the asynchronous list shall become known in the current shell execution environment; see Section 2.12 (on page 2263).

If the *wait* utility is invoked with no operands, it shall wait until all process IDs known to the invoking shell have terminated and exit with a zero exit status.

If one or more *pid* operands are specified that represent known process IDs, the *wait* utility shall wait until all of them have terminated. If one or more *pid* operands are specified that represent unknown process IDs, *wait* shall treat them as if they were known process IDs that exited with exit status 127. The exit status returned by the *wait* utility shall be the exit status of the process requested by the last *pid* operand.

The known process IDs are applicable only for invocations of *wait* in the current shell execution environment.

OPTIONS

None.

OPERANDS

The following operand shall be supported:

pid One of the following:

1. The unsigned decimal integer process ID of a command, for which the utility is to wait for the termination.
2. A job control job ID (see the Base Definitions volume of IEEE Std 1003.1-200x, Section 3.203, Job Control Job ID) 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 Section 2.12 (on page 2263). The exit status of *wait* shall be determined by the last command in the pipeline.

Note: The job control job ID type of *pid* is only available on systems supporting the User Portability Utilities option.

STDIN

Not used.

INPUT FILES

None.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *wait*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used only for diagnostic messages.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

If one or more operands were specified, all of them have terminated or were not known by the invoking shell, and the status of the last operand specified is known, then the exit status of *wait* shall be the exit status information of the command indicated by the last operand specified. If the process terminated abnormally due to the receipt of a signal, the exit status shall be greater than 128 and shall be distinct from the exit status generated by other signals, but the exact value is unspecified. (See the *kill* **–l** option.) Otherwise, the *wait* utility shall exit with one of the following values:

0 The *wait* utility was invoked with no operands and all process IDs known by the invoking shell have terminated.

1-126 The *wait* utility detected an error.

127 The command identified by the last *pid* operand specified is unknown.

CONSEQUENCES OF ERRORS

Default.

APPLICATION USAGE

On most implementations, *wait* is a shell built-in. If it is called in a subshell or separate utility execution environment, such as one of the following:

```
(wait)
nohup wait ...
find . –exec wait ... \;
```

it returns immediately because there are no known process IDs to wait for in those environments.

Historical implementations of interactive shells have discarded the exit status of terminated background processes before each shell prompt. Therefore, the status of background processes was usually lost unless it terminated while *wait* was waiting for it. This could be a serious problem when a job that was expected to run for a long time actually terminated quickly with a syntax or initialization error because the exit status returned was usually zero if the requested

process ID was not found. This volume of IEEE Std 1003.1-200x requires the implementation to keep the status of terminated jobs available until the status is requested, so that scripts like:

```
j1&
p1=$!
j2&
wait $p1
echo Job 1 exited with status $?
wait $!
echo Job 2 exited with status $?
```

works without losing status on any of the jobs. The shell is allowed to discard the status of any process that it determines the application cannot get the process ID from the shell. It is also required to remember only {CHILD_MAX} number of processes in this way. Since the only way to get the process ID from the shell is by using the '!' shell parameter, the shell is allowed to discard the status of an asynchronous list if "$!" was not referenced before another asynchronous list was started. (This means that the shell only has to keep the status of the last asynchronous list started if the application did not reference "$!". If the implementation of the shell is smart enough to determine that a reference to "$!" was not saved anywhere that the application can retrieve it later, it can use this information to trim the list of saved information. Note also that a successful call to *wait* with no operands discards the exit status of all asynchronous lists.)

If the exit status of *wait* is greater than 128, there is no way for the application to know if the waited-for process exited with that value or was killed by a signal. Since most utilities exit with small values, there is seldom any ambiguity. Even in the ambiguous cases, most applications just need to know that the asynchronous job failed; it does not matter whether it detected an error and failed or was killed and did not complete its job normally.

EXAMPLES

Although the exact value used when a process is terminated by a signal is unspecified, if it is known that a signal terminated a process, a script can still reliably figure out which signal using *kill* as shown by the following script:

```
sleep 1000&
pid=$!
kill –kill $pid
wait $pid
echo $pid was terminated by a SIG$(kill –l $?) signal.
```

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

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

either of the following commands returns the exit status of the second *sleep* in the pipeline:

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

RATIONALE

The description of *wait* does not refer to the *waitpid*() function from the System Interfaces volume of IEEE Std 1003.1-200x because that would needlessly overspecify this interface. However, the wording means that *wait* is required to wait for an explicit process when it is given an argument so that the status information of other processes is not consumed. Historical implementations use the *wait*() function defined in the System Interfaces volume of IEEE Std 1003.1-200x until *wait*() returns the requested process ID or finds that the requested

process does not exist. Because this means that a shell script could not reliably get the status of all background children if a second background job was ever started before the first job finished, it is recommended that the *wait* utility use a method such as the functionality provided by the *waitpid*( ) function.

The ability to wait for multiple *pid* operands was adopted from the KornShell.

This new functionality was added because it is needed to determine the exit status of any asynchronous list accurately. The only compatibility problem that this change creates is for a script like

```
while sleep 60 do
    job& echo Job started $(date) as $!  done
```

which causes the shell to monitor all of the jobs started until the script terminates or runs out of memory. This would not be a problem if the loop did not reference "$!" or if the script would occasionally *wait* for jobs it started.

FUTURE DIRECTIONS

None.

SEE ALSO

sh, the System Interfaces volume of IEEE Std 1003.1-200x, *waitpid*( )

CHANGE HISTORY

First released in Issue 2.

NAME

wc — word, line, and byte or character count

SYNOPSIS

```
wc [-c|-m][-lw][file...]
```

DESCRIPTION

The *wc* utility shall read one or more input files and, by default, write the number of <newline>s, words, and bytes contained in each input file to the standard output.

The utility also shall write a total count for all named files, if more than one input file is specified.

The *wc* utility shall consider a *word* to be a non-zero-length string of characters delimited by white space.

OPTIONS

The *wc* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section 12.2, Utility Syntax Guidelines.

The following options shall be supported:

-c Write to the standard output the number of bytes in each input file.

-l Write to the standard output the number of <newline>s in each input file.

-m Write to the standard output the number of characters in each input file.

-w Write to the standard output the number of words in each input file.

When any option is specified, *wc* shall report only the information requested by the specified options.

OPERANDS

The following operand shall be supported:

file A pathname of an input file. If no *file* operands are specified, the standard input shall be used.

STDIN

The standard input shall be used only if no *file* operands are specified. See the INPUT FILES section.

INPUT FILES

The input files may be of any type.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *wc*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and which characters are defined as white space characters.

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error and informative messages written to standard output.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

ASYNCHRONOUS EVENTS
Default.

STDOUT
By default, the standard output shall contain an entry for each input file of the form:

`"%d %d %d %s\n", <newlines>, <words>, <bytes>, <file>`

If the **–m** option is specified, the number of characters shall replace the <*bytes*> field in this format.

If any options are specified and the **–l** option is not specified, the number of <newline>s shall not be written.

If any options are specified and the **–w** option is not specified, the number of words shall not be written.

If any options are specified and neither **–c** nor **–m** is specified, the number of bytes or characters shall not be written.

If no input *file* operands are specified, no name shall be written and no <blank>s preceding the pathname shall be written.

If more than one input *file* operand is specified, an additional line shall be written, of the same format as the other lines, except that the word **total** (in the POSIX locale) shall be written instead of a pathname and the total of each column shall be written as appropriate. Such an additional line, if any, is written at the end of the output.

STDERR
The standard error shall be used only for diagnostic messages.

OUTPUT FILES
None.

EXTENDED DESCRIPTION
None.

EXIT STATUS
The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS
Default.

APPLICATION USAGE

The **–m** option is not a switch, but an option at the same level as **–c**. Thus, to produce the full default output with character counts instead of bytes, the command required is:

```
wc –mlw
```

EXAMPLES

None.

RATIONALE

The output file format pseudo-*printf*( ) string differs from the System V version of *wc*:

```
"%7d%7d%7d %s\n"
```

which produces possibly ambiguous and unparsable results for very large files, as it assumes no number shall exceed six digits.

Some historical implementations use only <space>, <tab>, and <newline> as word separators. The equivalent of the ISO C standard *isspace*( ) function is more appropriate.

The **–c** option stands for ‘‘character’’ count, even though it counts bytes. This stems from the sometimes erroneous historical view that bytes and characters are the same size. Due to international requirements, the **–m** option (reminiscent of ‘‘multi-byte’’) was added to obtain actual character counts.

Early proposals only specified the results when input files were text files. The current specification more closely matches historical practice. (Bytes, words, and <newline>s are counted separately and the results are written when an end-of-file is detected.)

Historical implementations of the *wc* utility only accepted one argument to specify the options **–c**, **–l**, and **–w**. Some of them also had multiple occurrences of an option cause the corresponding count to be written multiple times and had the order of specification of the options affect the order of the fields on output, but did not document either of these. Because common usage either specifies no options or only one option, and because none of this was documented, the changes required by this volume of IEEE Std 1003.1-200x should not break many historical applications (and do not break any historical conforming applications).

FUTURE DIRECTIONS

None.

SEE ALSO

cksum

CHANGE HISTORY

First released in Issue 2.

40038 **NAME**

40039 what — identify SCCS files (**DEVELOPMENT**)

40040 **SYNOPSIS**

40041 XSI
```
what [-s] file...
```
40042

40043 **DESCRIPTION**

40044 The *what* utility shall search the given files for all occurrences of the pattern that *get* (see *get* (on
40045 page 2675)) substitutes for the %**Z**% keyword ("@(#)") and shall write to standard output what
40046 follows until the first occurrence of one of the following:

40047 `"  >  newline  \  NUL`

40048 **OPTIONS**

40049 The *what* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
40050 12.2, Utility Syntax Guidelines.

40051 The following option shall be supported: |

40052 **-s** Quit after finding the first occurrence of the pattern in each file.

40053 **OPERANDS**

40054 The following operands shall be supported:

40055 *file* A pathname of a file to search.

40056 **STDIN**

40057 Not used.

40058 **INPUT FILES**

40059 The input files shall be of any file type. |

40060 **ENVIRONMENT VARIABLES**

40061 The following environment variables shall affect the execution of *what*:

40062 *LANG* Provide a default value for the internationalization variables that are unset or null.
40063 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
40064 Internationalization Variables for the precedence of internationalization variables
40065 used to determine the values of locale categories.)

40066 *LC_ALL* If set to a non-empty string value, override the values of all the other
40067 internationalization variables.

40068 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
40069 characters (for example, single-byte as opposed to multi-byte characters in
40070 arguments and input files).

40071 *LC_MESSAGES*
40072 Determine the locale that should be used to affect the format and contents of
40073 diagnostic messages written to standard error.

40074 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

40075 **ASYNCHRONOUS EVENTS**

40076 Default.

40077 **STDOUT**

40078 The standard output shall consist of the following for each *file* operand:

40079 `"%s:\n\t%s\n", <pathname>, <identification string>`

40080 **STDERR**

40081 The standard error shall be used only for diagnostic messages.

40082 **OUTPUT FILES**

40083 None.

40084 **EXTENDED DESCRIPTION**

40085 None.

40086 **EXIT STATUS**

40087 The following exit values shall be returned:

40088 0 Any matches were found.

40089 1 Otherwise.

40090 **CONSEQUENCES OF ERRORS**

40091 Default.

40092 **APPLICATION USAGE**

40093 The *what* utility is intended to be used in conjunction with the SCCS command *get*, which
40094 automatically inserts identifying information, but it can also be used where the information is
40095 inserted by any other means.

40096 When the string `"@(#)"` is included in a library routine in a shared library, it might not be found
40097 in an **a.out** file using that library routine.

40098 **EXAMPLES**

40099 If the C-language program in file **f.c** contains:

40100
```
char ident[] = "@(#)identification information";
```

40101 and **f.c** is compiled to yield **f.o** and **a.out**, then the command:

40102
```
what f.c f.o a.out
```

40103 writes:

40104
```
f.c:
    identification information
    ...
f.o:
    identification information
    ...
a.out:
    identification information
    ...
```

40113 **RATIONALE**

40114 None.

40115 **FUTURE DIRECTIONS**

40116 None.

40117 **SEE ALSO**

40118 *get*

40119 **CHANGE HISTORY**

40120 First released in Issue 2.

40121 **NAME**

40122 who — display who is on the system

40123 **SYNOPSIS**

40124 UP
```
who [–mTu]
```
40125

40126 XSI
```
who [–mu]–s[–bHlprt][file]
```

40127
```
who [–mTu][–abdHlprt][file]
```

40128
```
who –q [file]
```

40129
```
who am i
```

40130
```
who am I
```

40131

40132 **DESCRIPTION**

40133 The *who* utility shall list various pieces of information about accessible users. The domain of
40134 accessibility is implementation-defined.

40135 XSI Based on the options given, *who* can also list the user's name, terminal line, login time, elapsed
40136 time since activity occurred on the line, and the process ID of the command interpreter for each
40137 current system user.

40138 **OPTIONS**

40139 The *who* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
40140 12.2, Utility Syntax Guidelines.

40141 The following options shall be supported. The metavariables, such as <*line*>, refer to fields
40142 described in the STDOUT section.

40143 XSI **–a** Process the implementation-defined database or named file with the **–b**, **–d**, **–l**, **–p**,
40144 **–r**, **–t**, **–T** and **–u** options turned on.

40145 XSI **–b** Write the time and date of the last reboot.

40146 XSI **–d** Write a list of all processes that have expired and not been respawned by the *init* |
40147 system process. The <*exit*> field shall appear for dead processes and contain the |
40148 termination and exit values of the dead process. This can be useful in determining |
40149 why a process terminated. |

40150 XSI **–H** Write column headings above the regular output.

40151 XSI **–l** (The letter ell.) List only those lines on which the system is waiting for someone to |
40152 login. The <*name*> field shall be **LOGIN** in such cases. Other fields shall be the |
40153 same as for user entries except that the <*state*> field does not exist. |

40154 **–m** Output only information about the current terminal.

40155 XSI **–p** List any other process that is currently active and has been previously spawned by
40156 *init*.

40157 XSI **–q** (Quick.) List only the names and the number of users currently logged on. When |
40158 this option is used, all other options shall be ignored. |

40159 XSI **–r** Write the current *run-level* of the *init* process.

40160 XSI **–s** List only the <*name*>, <*line*>, and <*time*> fields. This is the default case.

40161 XSI **–t** Indicate the last change to the system clock.

40162 **–T** Show the state of each terminal, as described in the STDOUT section.

40163 **–u** Write ‘‘idle time’’ for each displayed user in addition to any other information. The
40164 idle time is the time since any activity occurred on the user’s terminal. The method
40165 XSI of determining this is unspecified. This option shall list only those users who are
40166 currently logged in. The <*name*> is the user’s login name. The <*line*> is the name of
40167 the line as found in the directory **/dev**. The <*time*> is the time that the user logged
40168 in. The <*activity*> is the number of hours and minutes since activity last occurred
40169 on that particular line. A dot indicates that the terminal has seen activity in the last
40170 minute and is therefore ‘‘current’’. If more than twenty-four hours have elapsed or
40171 the line has not been used since boot time, the entry shall be marked <*old*>. This
40172 field is useful when trying to determine whether a person is working at the
40173 terminal or not. The <*pid*> is the process ID of the user’s login process.

40174 **OPERANDS**

40175 XSI The following operands shall be supported:

40176 **am i**, **am I** In the POSIX locale, limit the output to describing the invoking user, equivalent to
40177 the **–m** option. The **am** and **i** or **I** must be separate arguments.

40178 *file* Specify a pathname of a file to substitute for the implementation-defined database
40179 of logged-on users that *who* uses by default.

40180 **STDIN**

40181 Not used.

40182 **INPUT FILES**

40183 None.

40184 **ENVIRONMENT VARIABLES**

40185 The following environment variables shall affect the execution of *who*:

40186 *LANG* Provide a default value for the internationalization variables that are unset or null.
40187 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
40188 Internationalization Variables for the precedence of internationalization variables
40189 used to determine the values of locale categories.)

40190 *LC_ALL* If set to a non-empty string value, override the values of all the other
40191 internationalization variables.

40192 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
40193 characters (for example, single-byte as opposed to multi-byte characters in
40194 arguments).

40195 *LC_MESSAGES*
40196 Determine the locale that should be used to affect the format and contents of
40197 diagnostic messages written to standard error.

40198 *LC_TIME* Determine the locale used for the format and contents of the date and time strings.

40199 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

40200 *TZ* Determine the timezone used when writing date and time information. If *TZ* is
40201 unset or null, an unspecified default timezone shall be used.

40202 **ASYNCHRONOUS EVENTS**

40203 Default.

40204 **STDOUT**

40205 The *who* utility shall write its default format to the standard output in an implementation-
40206 defined format, subject only to the requirement of containing the information described above.

40207 XSI OF 40208 XSI-conformant systems shall write the default information to the standard output in the following general format:

40209 `<name>[<state>]<line><time>[<activity>][<pid>][<comment>][<exit>]`

40210 The following format shall be used for the **–T** option:

40211 `"%s %c %s %s\n" <name>, <terminal state>, <terminal name>,`
40212 `<time of login>`

40213 where <*terminal state*> is one of the following characters:

40214 + The terminal allows write access to other users.

40215 – The terminal denies write access to other users.

40216 ? The terminal write-access state cannot be determined.

40217 In the POSIX locale, the <*time of login*> shall be equivalent in format to the output of:

40218 `date +"%b %e %H:%M"`

40219 If the **–u** option is used with **–T**, the idle time shall be added to the end of the previous format in
40220 an unspecified format.

40221 **STDERR**

40222 The standard error shall be used only for diagnostic messages.

40223 **OUTPUT FILES**

40224 None.

40225 **EXTENDED DESCRIPTION**

40226 None.

40227 **EXIT STATUS**

40228 The following exit values shall be returned:

40229 0 Successful completion.

40230 >0 An error occurred.

40231 **CONSEQUENCES OF ERRORS**

40232 Default.

40233 **APPLICATION USAGE**

40234 The name *init* used for the system process is the most commonly used on historical systems, but
40235 it may vary.

40236 The ‘‘domain of accessibility’’ referred to is a broad concept that permits interpretation either on
40237 a very secure basis or even to allow a network-wide implementation like the historical *rwho*.

40238 **EXAMPLES**

40239 None.

40240 **RATIONALE**

40241 Due to differences between historical implementations, the base options provided were a
40242 compromise to allow users to work with those functions. The standard developers also
40243 considered removing all the options, but felt that these options offered users valuable
40244 functionality. Additional options to match historical systems are available on XSI-conformant

systems.

It is recognized that the *who* command may be of limited usefulness, especially in a multi-level secure environment. The standard developers considered, however, that having some standard method of determining the ‘‘accessibility’’ of other users would aid user portability.

No format was specified for the default *who* output for systems not supporting the XSI Extension. In such a user-oriented command, designed only for human use, this was not considered to be a deficiency.

The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, and *write* require that they use the same format.

It is acceptable for an implementation to produce no output for an invocation of *who* **mil**.

FUTURE DIRECTIONS

None.

SEE ALSO

mesg

CHANGE HISTORY

First released in Issue 2.

Issue 6

This utility is now marked as part of the User Portability Utilities option.

The *TZ* entry is added to the ENVIRONMENT VARIABLES section.

40264 **NAME**

40265 write — write to another user

40266 **SYNOPSIS**

40267 UP
```
write user_name [terminal]
```
40268

40269 **DESCRIPTION**

40270 The *write* utility shall read lines from the user's standard input and write them to the terminal of
40271 another user. When first invoked, it shall write the message:

40272 **Message from** *sender-login-id* (*sending-terminal*) [*date*]...

40273 to *user_name*. When it has successfully completed the connection, the sender's terminal shall be
40274 alerted twice to indicate that what the sender is typing is being written to the recipient's
40275 terminal.

40276 If the recipient wants to reply, this can be accomplished by typing:

40277 `write` *sender-login-id* [*sending-terminal*]

40278 upon receipt of the initial message. Whenever a line of input as delimited by a NL, EOF, or EOL
40279 special character (see the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General
40280 Terminal Interface) is accumulated while in canonical input mode, the accumulated data shall be
40281 written on the other user's terminal. Characters shall be processed as follows:

40282 - Typing <alert> shall write the alert character to the recipient's terminal.

40283 - Typing the erase and kill characters shall affect the sender's terminal in the manner described
40284 by the **termios** interface in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11,
40285 General Terminal Interface.

40286 - Typing the interrupt or end-of-file characters shall cause *write* to write an appropriate
40287 message ("EOT\n" in the POSIX locale) to the recipient's terminal and exit.

40288 - Typing characters from *LC_CTYPE* classifications **print** or **space** shall cause those characters
40289 to be sent to the recipient's terminal.

40290 - When and only when the *stty* **iexten** local mode is enabled, the existence and processing of
40291 additional special control characters and multi-byte or single-byte functions is
40292 implementation-defined.

40293 - Typing other non-printable characters shall cause implementation-defined sequences of
40294 printable characters to be written to the recipient's terminal.

40295 To write to a user who is logged in more than once, the *terminal* argument can be used to indicate
40296 which terminal to write to; otherwise, the recipient's terminal is selected in an implementation-
40297 defined manner and an informational message is written to the sender's standard output,
40298 indicating which terminal was chosen.

40299 Permission to be a recipient of a *write* message can be denied or granted by use of the *mesg*
40300 utility. However, a user's privilege may further constrain the domain of accessibility of other
40301 users' terminals. The *write* utility shall fail when the user lacks the appropriate privileges to
40302 perform the requested action.

40303 **OPTIONS**

40304 None.

40305 **OPERANDS**

40306 The following operands shall be supported:

40307 *user_name* Login name of the person to whom the message shall be written. The application
40308 shall ensure that this operand is of the form returned by the *who* utility.

40309 *terminal* Terminal identification in the same format provided by the *who* utility.

40310 **STDIN**

40311 Lines to be copied to the recipient's terminal is read from standard input.

40312 **INPUT FILES**

40313 None.

40314 **ENVIRONMENT VARIABLES**

40315 The following environment variables shall affect the execution of *write*:

40316 *LANG* Provide a default value for the internationalization variables that are unset or null.
40317 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
40318 Internationalization Variables for the precedence of internationalization variables
40319 used to determine the values of locale categories.)

40320 *LC_ALL* If set to a non-empty string value, override the values of all the other
40321 internationalization variables.

40322 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
40323 characters (for example, single-byte as opposed to multi-byte characters in
40324 arguments and input files). If the recipient's locale does not use an *LC_CTYPE*
40325 equivalent to the sender's, the results are undefined.

40326 *LC_MESSAGES*
40327 Determine the locale that should be used to affect the format and contents of
40328 diagnostic messages written to standard error and informative messages written to
40329 standard output.

40330 XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

40331 **ASYNCHRONOUS EVENTS**

40332 If an interrupt signal is received, *write* shall write an appropriate message on the recipient's
40333 terminal and exits with a status of zero. It shall take the standard action for all other signals.

40334 **STDOUT**

40335 An informational message shall be written to standard output if a recipient is logged in more
40336 than once.

40337 **STDERR**

40338 The standard error shall be used only for diagnostic messages.

40339 **OUTPUT FILES**

40340 The recipient's terminal is used for output.

40341 **EXTENDED DESCRIPTION**

40342 None.

40343 **EXIT STATUS**

40344 The following exit values shall be returned:

40345 0 Successful completion.

40346 >0 The addressed user is not logged on or the addressed user denies permission.

40347 **CONSEQUENCES OF ERRORS**

40348 Default.

40349 **APPLICATION USAGE**

40350 The *talk* utility is considered by some users to be a more usable utility on full-screen terminals.

40351 **EXAMPLES**

40352 None.

40353 **RATIONALE**

40354 The *write* utility was included in this volume of IEEE Std 1003.1-200x since it can be
40355 implemented on all terminal types. The standard developers considered the *talk* utility, which
40356 cannot be implemented on certain terminals, to be a ‘‘better’’ communications interface. Both of
40357 these programs are in widespread use on historical implementations. Therefore, the standard
40358 developers decided that both utilities should be specified.

40359 The format of the terminal name is unspecified, but the descriptions of *ps*, *talk*, *who*, and *write*
40360 require that they all use or accept the same format.

40361 **FUTURE DIRECTIONS**

40362 None.

40363 **SEE ALSO**

40364 *mesg*, *talk*, *who*, the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 11, General
40365 Terminal Interface

40366 **CHANGE HISTORY**

40367 First released in Issue 2.

40368 **Issue 5**

40369 FUTURE DIRECTIONS section added.

40370 **Issue 6**

40371 This utility is now marked as part of the User Portability Utilities option.

40372 The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

40373 **NAME**

40374 xargs — construct argument lists and invoke utility

40375 **SYNOPSIS**

40376 XSI `xargs [–t][–p]][–E` *`eofstr`*`][–I` *`replstr`*`][–L` *`number`*`][–n` *`number`* `[–x]]`

40377 `[–s` *`size`*`][`*`utility`* `[`*`argument...`*`]]`

40378 **DESCRIPTION**

40379 The *xargs* utility shall construct a command line consisting of the *utility* and *argument* operands
40380 specified followed by as many arguments read in sequence from standard input as fit in length
40381 and number constraints specified by the options. The *xargs* utility shall then invoke the
40382 constructed command line and wait for its completion. This sequence shall be repeated until one
40383 of the following occurs:

40384 • An end-of-file condition is detected on standard input.

40385 • The logical end-of-file string (see the **–E** *eofstr* option) is found on standard input after
40386 double-quote processing, apostrophe processing, and backslash escape processing (see next
40387 paragraph).

40388 • An invocation of a constructed command line returns an exit status of 255.

40389 The application shall ensure that arguments in the standard input are separated by unquoted
40390 <blank>s, unescaped <blank>s or <newline>s. A string of zero or more non-double-quote (`'"'`)
40391 characters and non-<newline>s can be quoted by enclosing them in double-quotes. A string of
40392 zero or more non-apostrophe (`'\''`) characters and non-<newline>s can be quoted by enclosing
40393 them in apostrophes. Any unquoted character can be escaped by preceding it with a backslash.
40394 The utility named by *utility* shall be executed one or more times until the end-of-file is reached
40395 or the logical end-of file string is found. The results are unspecified if the utility named by *utility*
40396 attempts to read from its standard input.

40397 The generated command line length shall be the sum of the size in bytes of the utility name and
40398 each argument treated as strings, including a null byte terminator for each of these strings. The
40399 *xargs* utility shall limit the command line length such that when the command line is invoked,
40400 the combined argument and environment lists (see the *exec* family of functions in the System
40401 Interfaces volume of IEEE Std 1003.1-200x) shall not exceed {ARG_MAX}–2 048 bytes. Within
40402 this constraint, if neither the **–n** nor the **–s** option is specified, the default command line length
40403 shall be at least {LINE_MAX}.

40404 **OPTIONS**

40405 The *xargs* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
40406 12.2, Utility Syntax Guidelines.

40407 The following options shall be supported:

40408 **–E** *eofstr* Use *eofstr* as the logical end-of-file string. If **–E** is not specified, it is unspecified
40409 whether the logical end-of-file string is the underscore character (`'_'`) or the end-
40410 of-file string capability is disabled. When *eofstr* is the null string, the logical end-
40411 of-file string capability shall be disabled and underscore characters shall be taken
40412 literally.

40413 XSI **–I** *replstr* Insert mode: *utility* is executed for each line from standard input, taking the entire
40414 line as a single argument, inserting it in *argument*s for each occurrence of *replstr*. A
40415 maximum of five arguments in *argument*s can each contain one or more instances
40416 of *replstr*. Any <blank>s at the beginning of each line shall be ignored.
40417 Constructed arguments cannot grow larger than 255 bytes. Option **–x** shall be
40418 forced on.

40419 XSI **–L** *number* The *utility* shall be executed for each non-empty *number* lines of arguments from
40420 standard input. The last invocation of *utility* shall be with fewer lines of arguments
40421 if fewer than *number* remain. A line is considered to end with the first <newline>
40422 unless the last character of the line is a <blank>; a trailing <blank> signals
40423 continuation to the next non-empty line, inclusive. The **–L** and **–n** options are
40424 mutually-exclusive; the last one specified shall take effect.

40425 **–n** *number* Invoke *utility* using as many standard input arguments as possible, up to *number* (a
40426 positive decimal integer) arguments maximum. Fewer arguments shall be used if:

40427 • The command line length accumulated exceeds the size specified by the **–s**
40428 option (or {LINE_MAX} if there is no **–s** option).

40429 • The last iteration has fewer than but not zero, operands remaining.

40430 **–p** Prompt mode: the user is asked whether to execute *utility* at each invocation. Trace
40431 mode (**–t**) is turned on to write the command instance to be executed, followed by
40432 a prompt to standard error. An affirmative response read from **/dev/tty** shall
40433 execute the command; otherwise, that particular invocation of *utility* shall be
40434 skipped.

40435 **–s** *size* Invoke *utility* using as many standard input arguments as possible yielding a
40436 command line length less than *size* (a positive decimal integer) bytes. Fewer
40437 arguments shall be used if:

40438 • The total number of arguments exceeds that specified by the **–n** option.

40439 XSI • The total number of lines exceeds that specified by the **–L** option.

40440 • End-of-file is encountered on standard input before *size* bytes are accumulated.

40441 Values of *size* up to at least {LINE_MAX} bytes shall be supported, provided that
40442 the constraints specified in the DESCRIPTION are met. It shall not be considered
40443 an error if a value larger than that supported by the implementation or exceeding
40444 the constraints specified in the DESCRIPTION is given; *xargs* shall use the largest
40445 value it supports within the constraints.

40446 **–t** Enable trace mode. Each generated command line shall be written to standard
40447 error just prior to invocation.

40448 **–x** Terminate if a command line containing *number* arguments (see the **–n** option
40449 XSI above) or *number* lines (see the **–L** option above) will not fit in the implied or
40450 specified size (see the **–s** option above).

40451 **OPERANDS**

40452 The following operands shall be supported:

40453 *utility* The name of the utility to be invoked, found by search path using the *PATH*
40454 environment variable, described in the Base Definitions volume of
40455 IEEE Std 1003.1-200x, Chapter 8, Environment Variables. If *utility* is omitted, the
40456 default shall be the *echo* utility. If the *utility* operand names any of the special
40457 built-in utilities in Section 2.14 (on page 2266), the results are undefined.

40458 *argument* An initial option or operand for the invocation of *utility*.

40459 **STDIN**

40460 The standard input shall be a text file. The results are unspecified if an end-of-file condition is
40461 detected immediately following an escaped <newline>.

INPUT FILES

The file **/dev/tty** shall be used to read responses required by the –**p** option.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *xargs*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_COLLATE
Determine the locale for the behavior of ranges, equivalence classes and multi-character collating elements used in the extended regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES* category.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files) and the behavior of character classes used in the extended regular expression defined for the **yesexpr** locale keyword in the *LC_MESSAGES* category.

LC_MESSAGES
Determine the locale for the processing of affirmative responses and that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

PATH Determine the location of *utility*, as described in the Base Definitions volume of IEEE Std 1003.1-200x, Chapter 8, Environment Variables.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

The standard error shall be used for diagnostic messages and the –**t** and –**p** options. If the –**t** option is specified, the *utility* and its constructed argument list shall be written to standard error, as it will be invoked, prior to invocation. If –**p** is specified, a prompt of the following format shall be written (in the POSIX locale):

```
"?..."
```

at the end of the line of the output from –**t**.

OUTPUT FILES

None.

EXTENDED DESCRIPTION

None.

EXIT STATUS

The following exit values shall be returned:

0 All invocations of *utility* returned exit status zero.

1-125 A command line meeting the specified requirements could not be assembled, one or more of the invocations of *utility* returned a non-zero exit status, or some other error occurred.

126 The utility specified by *utility* was found but could not be invoked.

127 The utility specified by *utility* could not be found.

CONSEQUENCES OF ERRORS

If a command line meeting the specified requirements cannot be assembled, the utility cannot be invoked, an invocation of the utility is terminated by a signal, or an invocation of the utility exits with exit status 255, the *xargs* utility shall write a diagnostic message and exit without processing any remaining input.

APPLICATION USAGE

The 255 exit status allows a utility being used by *xargs* to tell *xargs* to terminate if it knows no further invocations using the current data stream will succeed. Thus, *utility* should explicitly *exit* with an appropriate value to avoid accidentally returning with 255.

Note that input is parsed as lines; <blank>s separate arguments. If *xargs* is used to bundle output of commands like *find dir* **–print** or *ls* into commands to be executed, unexpected results are likely if any filenames contain any <blank>s or <newline>s. This can be fixed by using *find* to call a script that converts each file found into a quoted string that is then piped to *xargs*. Note that the quoting rules used by *xargs* are not the same as in the shell. They were not made consistent here because existing applications depend on the current rules and the shell syntax is not fully compatible with it. An easy rule that can be used to transform any string into a quoted form that *xargs* interprets correctly is to precede each character in the string with a backslash.

On implementations with a large value for {ARG_MAX}, *xargs* may produce command lines longer than {LINE_MAX}. For invocation of utilities, this is not a problem. If *xargs* is being used to create a text file, users should explicitly set the maximum command line length with the **–s** option.

The *command*, *env*, *nice*, *nohup*, *time*, and *xargs* utilities have been specified to use exit code 127 if an error occurs so that applications can distinguish ‘‘failure to find a utility’’ from ‘‘invoked utility exited with an error indication’’. The value 127 was chosen because it is not commonly used for other meanings; most utilities use small values for ‘‘normal error conditions’’ and the values above 128 can be confused with termination due to receipt of a signal. The value 126 was chosen in a similar manner to indicate that the utility could be found, but not invoked. Some scripts produce meaningful error messages differentiating the 126 and 127 cases. The distinction between exit codes 126 and 127 is based on KornShell practice that uses 127 when all attempts to *exec* the utility fail with [ENOENT], and uses 126 when any attempt to *exec* the utility fails for any other reason.

EXAMPLES

1. The following command combines the output of the parenthesised commands onto one line, which is then written to the end-of-file **log**:

```
(logname; date; printf "%s\n" "$0 $*") | xargs >>log
```

2. The following command invokes *diff* with successive pairs of arguments originally typed as command line arguments (assuming there are no embedded <blank>s in the elements of the original argument list):

```
printf "%s\n" "$*" | xargs –n 2 –x diff
```

3. In the following commands, the user is asked which files in the current directory are to be archived. The files are archived into **arch**; *a*, one at a time, or *b*, many at a time.

   ```
   a. ls | xargs –p –L 1 ar –r arch
   ```

   ```
   b. ls | xargs –p –L 1 | xargs ar –r arch
   ```

4. The following executes with successive pairs of arguments originally typed as command line arguments:

   ```
   echo $* | xargs –n 2 diff
   ```

5. On XSI-conformant systems, the following moves all files from directory **$1** to directory **$2**, and echoes each move command just before doing it:

   ```
   ls $1 | xargs –I {} –t mv $1/{} $2/{}
   ```

RATIONALE

The *xargs* utility was usually found only in System V-based systems; BSD systems included an *apply* utility that provided functionality similar to *xargs* –**n** *number*. The SVID lists *xargs* as a software development extension. This volume of IEEE Std 1003.1-200x does not share the view that it is used only for development, and therefore it is not optional.

The classic application of the *xargs* utility is in conjunction with the *find* utility to reduce the number of processes launched by a simplistic use of the *find* –**exec** combination. The *xargs* utility is also used to enforce an upper limit on memory required to launch a process. With this basis in mind, this volume of IEEE Std 1003.1-200x selected only the minimal features required.

Although the 255 exit status is mostly an accident of historical implementations, it allows a utility being used by *xargs* to tell *xargs* to terminate if it knows no further invocations using the current data stream shall succeed. Any non-zero exit status from a utility falls into the 1-125 range when *xargs* exits. There is no statement of how the various non-zero utility exit status codes are accumulated by *xargs*. The value could be the addition of all codes, their highest value, the last one received, or a single value such as 1. Since no algorithm is arguably better than the others, and since many of the standard utilities say little more (portably) than ‘‘pass/fail’’, no new algorithm was invented.

Several other *xargs* options were withdrawn because simple alternatives already exist within this volume of IEEE Std 1003.1-200x. For example, the –**i** *replstr* option can be just as efficiently performed using a shell *for* loop. Since *xargs* calls an *exec* function with each input line, the –**i** option does not usually exploit the grouping capabilities of *xargs*.

The requirement that *xargs* never produce command lines such that invocation of *utility* is within 2 048 bytes of hitting the POSIX *exec* {ARG_MAX} limitations is intended to guarantee that the invoked utility has room to modify its environment variables and command line arguments and still be able to invoke another utility. Note that the minimum {ARG_MAX} allowed by the System Interfaces volume of IEEE Std 1003.1-200x is 4 096 bytes and the minimum value allowed by the this volume of IEEE Std 1003.1-200x is 2 048 bytes; therefore, the 2 048 bytes difference seems reasonable. Note, however, that *xargs* may never be able to invoke a utility if the environment passed in to *xargs* comes close to using {ARG_MAX} bytes.

The version of *xargs* required by this volume of IEEE Std 1003.1-200x is required to wait for the completion of the invoked command before invoking another command. This was done because historical scripts using *xargs* assumed sequential execution. Implementations wanting to provide parallel operation of the invoked utilities are encouraged to add an option enabling parallel invocation, but should still wait for termination of all of the children before *xargs* terminates normally.

The **–e** option was omitted from the ISO POSIX-2:1993 standard in the belief that the *eofstr* option-argument was recognized only when it was on a line by itself and before quote and escape processing were performed, and that the logical end-of-file processing was only enabled if a **–e** option was specified. In that case, a simple *sed* script could be used to duplicate the **–e** functionality. Further investigation revealed that:

- The logical end-of-file string was checked for after quote and escape processing, making a *sed* script that provided equivalent functionality much more difficult to write.
- The default was to perform logical end-of-file processing with an underscore as the logical end-of-file string.

To correct this misunderstanding, the **–E** *eofstr* option was adopted from the X/Open Portability Guide. Users should note that the description of the **–E** option matches historical documentation of the **–e** option (which was not adopted because it did not support the Utility Syntax Guidelines), by saying that if *eofstr* is the null string, logical end-of-file processing is disabled. Historical implementations of *xargs* actually did not disable logical end-of-file processing; they treated a null argument found in the input as a logical end-of-file string. (A null *string* argument could be generated using single or double quotes (`' '` or `" "`). Since this behavior was not documented historically, it is considered to be a bug.

FUTURE DIRECTIONS

None.

SEE ALSO

echo

CHANGE HISTORY

First released in Issue 2.

Issue 5

Second FUTURE DIRECTION added.

Issue 6

The obsolescent **–e**, **–i**, and **–l** options are removed.

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The **–p** option is added.
- In the INPUT FILES section, the file **/dev/tty** is used to read responses required by the **–p** option.
- The STDERR section is updated to describe the **–p** option.

The description of the **–E** option is aligned with the ISO POSIX-2:1993 standard.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

40630 **NAME**

40631 yacc — yet another compiler compiler (**DEVELOPMENT**)

40632 **SYNOPSIS**

40633 CD
```
yacc [–dltv][–b file_prefix][–p sym_prefix] grammar
```
40634

40635 **DESCRIPTION**

40636 The *yacc* utility shall read a description of a context-free grammar in *file* and write C source code,
40637 conforming to the ISO C standard, to a code file, and optionally header information into a
40638 header file, in the current directory. The C code shall define a function and related routines and
40639 macros for an automaton that executes a parsing algorithm meeting the requirements in
40640 **Algorithms** (on page 3272).

40641 The form and meaning of the grammar are described in the EXTENDED DESCRIPTION section.

40642 The C source code and header file shall be produced in a form suitable as input for the C
40643 compiler (see *c99* (on page 2413)).

40644 **OPTIONS**

40645 The *yacc* utility shall conform to the Base Definitions volume of IEEE Std 1003.1-200x, Section
40646 12.2, Utility Syntax Guidelines.

40647 The following options shall be supported:

40648 **–b** *file_prefix* Use *file_prefix* instead of **y** as the prefix for all output filenames. The code file
40649 **y.tab.c**, the header file **y.tab.h** (created when **–d** is specified), and the description
40650 file **y.output** (created when **–v** is specified), shall be changed to *file_prefix***.tab.c**,
40651 *file_prefix***.tab.h**, and *file_prefix***.output**, respectively.

40652 **–d** Write the header file; by default only the code file is written. The **#define**
40653 statements that associate the token codes assigned by *yacc* with the user-declared
40654 token names. This allows source files other than **y.tab.c** to access the token codes.

40655 **–l** Produce a code file that does not contain any **#line** constructs. If this option is not
40656 present, it is unspecified whether the code file or header file contains **#line**
40657 directives. This should only be used after the grammar and the associated actions
40658 are fully debugged.

40659 **–p** *sym_prefix* Use *sym_prefix* instead of **yy** as the prefix for all external names produced by *yacc*.
40660 The names affected shall include the functions *yyparse*( ), *yylex*( ), and *yyerror*( ),
40661 and the variables *yylval*, *yychar*, and *yydebug*. (In the remainder of this section, the
40662 six symbols cited are referenced using their default names only as a notational
40663 convenience.) Local names may also be affected by the **–p** option; however, the **–p**
40664 option shall not affect **#define** symbols generated by *yacc*.

40665 **–t** Modify conditional compilation directives to permit compilation of debugging
40666 code in the code file. Runtime debugging statements shall always be contained in
40667 the code file, but by default conditional compilation directives prevent their
40668 compilation.

40669 **–v** Write a file containing a description of the parser and a report of conflicts
40670 generated by ambiguities in the grammar.

40671 **OPERANDS**

40672 The following operand is required:

40673 *grammar* A pathname of a file containing instructions, hereafter called *grammar*, for which a
40674 parser is to be created. The format for the grammar is described in the EXTENDED

DESCRIPTION section.

STDIN

Not used.

INPUT FILES

The file *grammar* shall be a text file formatted as specified in the EXTENDED DESCRIPTION section.

ENVIRONMENT VARIABLES

The following environment variables shall affect the execution of *yacc*:

LANG Provide a default value for the internationalization variables that are unset or null. (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2, Internationalization Variables for the precedence of internationalization variables used to determine the values of locale categories.)

LC_ALL If set to a non-empty string value, override the values of all the other internationalization variables.

LC_CTYPE Determine the locale for the interpretation of sequences of bytes of text data as characters (for example, single-byte as opposed to multi-byte characters in arguments and input files).

LC_MESSAGES
Determine the locale that should be used to affect the format and contents of diagnostic messages written to standard error.

XSI *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

The *LANG* and *LC_** variables affect the execution of the *yacc* utility as stated. The *main*() function defined in **Yacc Library** (on page 3272) shall call:

```
setlocale(LC_ALL, "")
```

and thus, the program generated by *yacc* also shall be affected by the contents of these variables at runtime.

ASYNCHRONOUS EVENTS

Default.

STDOUT

Not used.

STDERR

If shift/reduce or reduce/reduce conflicts are detected in *grammar*, *yacc* shall write a report of those conflicts to the standard error in an unspecified format.

Standard error shall also be used for diagnostic messages.

OUTPUT FILES

The code file, the header file, and the description file shall be text files. All are described in the following sections.

Code File

This file shall contain the C source code for the *yyparse*( ) function. It shall contain code for the various semantic actions with macro substitution performed on them as described in the EXTENDED DESCRIPTION section. It also shall contain a copy of the **#define** statements in the header file. If a **%union** declaration is used, the declaration for YYSTYPE shall be also included in this file.

Header File

The header file shall contain **#define** statements that associate the token numbers with the token names. This allows source files other than the code file to access the token codes. If a **%union** declaration is used, the declaration for YYSTYPE and an *extern YYSTYPE yylval* declaration shall be also included in this file.

Description File

The description file shall be a text file containing a description of the state machine corresponding to the parser, using an unspecified format. Limits for internal tables (see **Limits** (on page 3273)) shall also be reported, in an implementation-defined manner. (Some implementations may use dynamic allocation techniques and have no specific limit values to report.)

EXTENDED DESCRIPTION

The *yacc* command accepts a language that is used to define a grammar for a target language to be parsed by the tables and code generated by *yacc*. The language accepted by *yacc* as a grammar for the target language is described below using the *yacc* input language itself.

The input *grammar* includes rules describing the input structure of the target language and code to be invoked when these rules are recognized to provide the associated semantic action. The code to be executed shall appear as bodies of text that are intended to be C-language code. The C-language inclusions are presumed to form a correct function when processed by *yacc* into its output files. The code included in this way shall be executed during the recognition of the target language.

Given a grammar, the *yacc* utility generates the files described in the OUTPUT FILES section. The code file can be compiled and linked using *c99*. If the declaration and programs sections of the grammar file did not include definitions of *main*( ), *yylex*( ), and *yyerror*( ), the compiled output requires linking with externally supplied version of those functions. Default versions of *main*( ) and *yyerror*( ) are supplied in the *yacc* library and can be linked in by using the **–l y** operand to *c99*. The *yacc* library interfaces need not support interfaces with other than the default **yy** symbol prefix. The application provides the lexical analyzer function, *yylex*( ); the *lex* utility is specifically designed to generate such a routine.

Input Language

The application shall ensure that every specification file consists of three sections in order: *declarations*, *grammar rules*, and *programs*, separated by double percent signs ("%%"). The declarations and programs sections can be empty. If the latter is empty, the preceding "%%" mark separating it from the rules section can be omitted.

The input is free form text following the structure of the grammar defined below.

40753 **Lexical Structure of the Grammar**

40754 The <blank>s, <newline>s, and <form-feed>s shall be ignored, except that the application shall
40755 ensure that they do not appear in names or multi-character reserved symbols. Comments shall
40756 be enclosed in "/* ... */", and can appear wherever a name is valid.

40757 Names are of arbitrary length, made up of letters, periods ('.'), underscores ('_'), and non-
40758 initial digits. Uppercase and lowercase letters are distinct. Conforming applications shall not |
40759 use names beginning in **yy** or **YY** since the *yacc* parser uses such names. Many of the names |
40760 appear in the final output of *yacc*, and thus they should be chosen to conform with any
40761 additional rules created by the C compiler to be used. In particular they appear in **#define**
40762 statements.

40763 A literal shall consist of a single character enclosed in single-quotes ('\''). All of the escape
40764 sequences supported for character constants by the ISO C standard shall be supported by *yacc*.

40765 The relationship with the lexical analyzer is discussed in detail below.

40766 The application shall ensure that the NUL character is not used in grammar rules or literals.

40767 **Declarations Section**

40768 The declarations section is used to define the symbols used to define the target language and
40769 their relationship with each other. In particular, much of the additional information required to
40770 resolve ambiguities in the context-free grammar for the target language is provided here.

40771 Usually *yacc* assigns the relationship between the symbolic names it generates and their
40772 underlying numeric value. The declarations section makes it possible to control the assignment
40773 of these values.

40774 It is also possible to keep semantic information associated with the tokens currently on the parse
40775 stack in a user-defined C-language **union**, if the members of the union are associated with the
40776 various names in the grammar. The declarations section provides for this as well.

40777 The first group of declarators below all take a list of names as arguments. That list can optionally
40778 be preceded by the name of a C union member (called a *tag* below) appearing within '<' and
40779 '>'. (As an exception to the typographical conventions of the rest of this volume of
40780 IEEE Std 1003.1-200x, in this case <*tag*> does not represent a metavariable, but the literal angle
40781 bracket characters surrounding a symbol.) The use of *tag* specifies that the tokens named on this
40782 line shall be of the same C type as the union member referenced by *tag*. This is discussed in
40783 more detail below.

40784 For lists used to define tokens, the first appearance of a given token can be followed by a
40785 positive integer (as a string of decimal digits). If this is done, the underlying value assigned to it
40786 for lexical purposes shall be taken to be that number. |

40787 The following declares *name* to be a token:

40788 `token [<`*tag*`>] `*name*` [`*number*`][`*name*` [`*number*`]]...`

40789 If *tag* is present, the C type for all tokens on this line shall be declared to be the type referenced
40790 by *tag*. If a positive integer, *number*, follows a *name*, that value shall be assigned to the token.

40791 The following declares *name* to be a token, and assigns precedence to it:

40792 `%left [<`*tag*`>] `*name*` [`*number*`][`*name*` [`*number*`]]...`
40793 `%right [<`*tag*`>] `*name*` [`*number*`][`*name*` [`*number*`]]...`

40794 One or more lines, each beginning with one of these symbols, can appear in this section. All
40795 tokens on the same line have the same precedence level and associativity; the lines are in order

of increasing precedence or binding strength. **%left** denotes that the operators on that line are left associative, and **%right** similarly denotes right associative operators. If *tag* is present, it shall declare a C type for *name*s as described for **%token**.

The following declares *name* to be a token, and indicates that this cannot be used associatively:

```
%nonassoc [<tag>] name [number][name [number]]...
```

If the parser encounters associative use of this token it reports an error. If *tag* is present, it shall declare a C type for *name*s as described for **%token**.

The following declares that union member *name*s are non-terminals, and thus it is required to have a *tag* field at its beginning:

```
%type <tag> name...
```

Because it deals with non-terminals only, assigning a token number or using a literal is also prohibited. If this construct is present, *yacc* shall perform type checking; if this construct is not present, the parse stack shall hold only the **int** type.

Every name used in *grammar* not defined by a **%token**, **%left**, **%right**, or **%nonassoc** declaration is assumed to represent a non-terminal symbol. The *yacc* utility shall report an error for any non-terminal symbol that does not appear on the left side of at least one grammar rule.

Once the type, precedence, or token number of a name is specified, it shall not be changed. If the first declaration of a token does not assign a token number, *yacc* shall assign a token number. Once this assignment is made, the token number shall not be changed by explicit assignment.

The following declarators do not follow the previous pattern.

The following declares the non-terminal *name* to be the *start symbol*, which represents the largest, most general structure described by the grammar rules:

```
%start name
```

By default, it is the left-hand side of the first grammar rule; this default can be overridden with this declaration.

The following declares the *yacc* value stack to be a union of the various types of values desired:

```
%union { body of union (in C) }
```

By default, the values returned by actions (see below) and the lexical analyzer shall be of type **int**. The *yacc* utility keeps track of types, and it shall insert corresponding union member names in order to perform strict type checking of the resulting parser.

Alternatively, given that at least one <*tag*> construct is used, the union can be declared in a header file (which shall be included in the declarations section by using an **#include** construct within **%{** and **%}**), and a **typedef** used to define the symbol YYSTYPE to represent this union. The effect of **%union** is to provide the declaration of YYSTYPE directly from the *yacc* input.

C-language declarations and definitions can appear in the declarations section, enclosed by the following marks:

```
%{ ... %}
```

These statements shall be copied into the code file, and have global scope within it so that they can be used in the rules and program sections.

The application shall ensure that the declarations section is terminated by the token **%%**.

40836 **Grammar Rules in yacc**

40837 The rules section defines the context-free grammar to be accepted by the function *yacc* generates,
40838 and associates with those rules C-language actions and additional precedence information. The
40839 grammar is described below, and a formal definition follows.

40840 The rules section is comprised of one or more grammar rules. A grammar rule has the form:

40841 `A : BODY ;`

40842 The symbol **A** represents a non-terminal name, and **BODY** represents a sequence of zero or
40843 more *name*s, *literal*s, and *semantic action*s that can then be followed by optional *precedence rule*s.
40844 Only the names and literals participate in the formation of the grammar; the semantic actions
40845 and precedence rules are used in other ways. The colon and the semicolon are *yacc* punctuation.
40846 If there are several successive grammar rules with the same left-hand side, the vertical bar `'|'`
40847 can be used to avoid rewriting the left-hand side; in this case the semicolon appears only after
40848 the last rule. The BODY part can be empty (or empty of names and literals) to indicate that the
40849 non-terminal symbol matches the empty string.

40850 The *yacc* utility assigns a unique number to each rule. Rules using the vertical bar notation are
40851 distinct rules. The number assigned to the rule appears in the description file.

40852 The elements comprising a BODY are:

40853 *name, literal* These form the rules of the grammar: *name* is either a *token* or a *non-terminal*; *literal*
40854 stands for itself (less the lexically required quotation marks).

40855 *semantic action*

40856 With each grammar rule, the user can associate actions to be performed each time
40857 the rule is recognized in the input process. (Note that the word ‘‘action’’ can also
40858 refer to the actions of the parser—shift, reduce, and so on.)

40859 These actions can return values and can obtain the values returned by previous
40860 actions. These values are kept in objects of type YYSTYPE (see **%union**). The
40861 result value of the action shall be kept on the parse stack with the left-hand side of
40862 the rule, to be accessed by other reductions as part of their right-hand side. By
40863 using the <*tag*> information provided in the declarations section, the code
40864 generated by *yacc* can be strictly type checked and contain arbitrary information. In
40865 addition, the lexical analyzer can provide the same kinds of values for tokens, if
40866 desired.

40867 An action is an arbitrary C statement and as such can do input or output, call
40868 subprograms and alter external variables. An action is one or more C statements
40869 enclosed in curly braces `'{'` and `'}'`.

40870 Certain pseudo-variables can be used in the action. These are macros for access to
40871 data structures known internally to *yacc*.

40872 $$ The value of the action can be set by assigning it to $$. If type
40873 checking is enabled and the type of the value to be assigned cannot
40874 be determined, a diagnostic message may be generated.

40875 $*number* This refers to the value returned by the component specified by the
40876 token *number* in the right side of a rule, reading from left to right;
40877 *number* can be zero or negative. If it is, it refers to the data associated
40878 with the name on the parser’s stack preceding the leftmost symbol of
40879 the current rule. (That is, `"$0"` refers to the name immediately
40880 preceding the leftmost name in the current rule, to be found on the
40881 parser’s stack and `"$–1"` refers to the symbol to *its* left.) If *number*

refers to an element past the current point in the rule, or beyond the bottom of the stack, the result is undefined. If type checking is enabled and the type of the value to be assigned cannot be determined, a diagnostic message may be generated.

$<*tag*>*number*
: These correspond exactly to the corresponding symbols without the *tag* inclusion, but allow for strict type checking (and preclude unwanted type conversions). The effect is that the macro is expanded to use *tag* to select an element from the YYSTYPE union (using *dataname.tag*). This is particularly useful if *number* is not positive.

$<*tag*>$
: This imposes on the reference the type of the union member referenced by *tag*. This construction is applicable when a reference to a left context value occurs in the grammar, and provides *yacc* with a means for selecting a type.

Actions can occur anywhere in a rule (not just at the end); an action can access values returned by actions to its left, and in turn the value it returns can be accessed by actions to its right. An action appearing in the middle of a rule shall be equivalent to replacing the action with a new non-terminal symbol and adding an empty rule with that non-terminal symbol on the left-hand side. The semantic action associated with the new rule shall be equivalent to the original action. The use of actions within rules might introduce conflicts that would not otherwise exist.

By default, the value of a rule shall be the value of the first element in it. If the first element does not have a type (particularly in the case of a literal) and type checking is turned on by **%type** an error message shall result.

precedence
: The keyword **%prec** can be used to change the precedence level associated with a particular grammar rule. Examples of this are in cases where a unary and binary operator have the same symbolic representation, but need to be given different precedences, or where the handling of an ambiguous if-else construction is necessary. The reserved symbol **%prec** can appear immediately after the body of the grammar rule and can be followed by a token name or a literal. It shall cause the precedence of the grammar rule to become that of the following token name or literal. The action for the rule as a whole can follow **%prec**.

If a program section follows, the application shall ensure that the grammar rules are terminated by **%%**.

Programs Section

The *programs* section can include the definition of the lexical analyzer *yylex*(), and any other functions, for example those used in the actions specified in the grammar rules. It is unspecified whether the programs section precedes or follows the semantic actions in the output file; therefore, if the application contains any macro definitions and declarations intended to apply to the code in the semantic actions, it shall place them within "%{ ... %}" in the declarations section.

Input Grammar

The following input to *yacc* yields a parser for the input to *yacc*. This formal syntax takes precedence over the preceding text syntax description.

The lexical structure is defined less precisely; **Lexical Structure of the Grammar** (on page 3264) defines most terms. The correspondence between the previous terms and the tokens below is as follows.

IDENTIFIER This corresponds to the concept of *name*, given previously. It also includes literals as defined previously.

C_IDENTIFIER This is a name, and additionally it is known to be followed by a colon. A literal cannot yield this token.

NUMBER A string of digits (a non-negative decimal integer).

TYPE, **LEFT**, **MARK**, **LCURL**, **RCURL**
These correspond directly to **%type**, **%left**, **%%**, **%{**, and **%}**.

{...} This indicates C-language source code, with the possible inclusion of '$' macros as discussed previously.

```
/* Grammar for the input to yacc. */
/* Basic entries. */
/* The following are recognized by the lexical analyzer. */

%token    IDENTIFIER      /* Includes identifiers and literals */
%token    C_IDENTIFIER    /* identifier (but not literal)
                             followed by a :. */
%token    NUMBER          /* [0-9][0-9]* */

/* Reserved words : %type=>TYPE %left=>LEFT, and so on */

%token    LEFT RIGHT NONASSOC TOKEN PREC TYPE START UNION

%token    MARK            /* The %% mark. */
%token    LCURL           /* The %{ mark. */
%token    RCURL           /* The %} mark. */

/* 8-bit character literals stand for themselves; */
/* tokens have to be defined for multi-byte characters. */

%start    spec

%%

spec  : defs MARK rules tail
      ;
tail  : MARK
      {
        /* In this action, set up the rest of the file. */
      }
      | /* Empty; the second MARK is optional. */
      ;
defs  : /* Empty. */
      |    defs def
      ;
def   : START IDENTIFIER
      |    UNION
```

```
40968               {
40969                 /* Copy union definition to output. */
40970               }
40971               |    LCURL
40972               {
40973                 /* Copy C code to output file. */
40974               }
40975                 RCURL
40976               |    rword tag nlist
40977               ;
40978       rword : TOKEN
40979               | LEFT
40980               | RIGHT
40981               | NONASSOC
40982               | TYPE
40983               ;
40984       tag   : /* Empty: union tag ID optional. */
40985               | '<' IDENTIFIER '>'
40986               ;
40987       nlist : nmno
40988               | nlist nmno
40989               ;
40990       nmno  : IDENTIFIER         /* Note: literal invalid with % type. */
40991               | IDENTIFIER NUMBER  /* Note: invalid with % type. */
40992               ;
40993       /* Rule section */
40994       rules : C_IDENTIFIER rbody prec
40995               | rules  rule
40996               ;
40997       rule  : C_IDENTIFIER rbody prec
40998               | '|' rbody prec
40999               ;
41000       rbody : /* empty */
41001               | rbody IDENTIFIER
41002               | rbody act
41003               ;
41004       act   : '{'
41005                 {
41006                   /* Copy action, translate $$, and so on. */
41007                 }
41008                 '}'
41009               ;
41010       prec  : /* Empty */
41011               | PREC IDENTIFIER
41012               | PREC IDENTIFIER act
41013               | prec ';'
41014               ;
```

Conflicts

The parser produced for an input grammar may contain states in which conflicts occur. The conflicts occur because the grammar is not LALR(1). An ambiguous grammar always contains at least one LALR(1) conflict. The *yacc* utility shall resolve all conflicts, using either default rules or user-specified precedence rules.

Conflicts are either shift/reduce conflicts or reduce/reduce conflicts. A shift/reduce conflict is where, for a given state and lookahead symbol, both a shift action and a reduce action are possible. A reduce/reduce conflict is where, for a given state and lookahead symbol, reductions by two different rules are possible.

The rules below describe how to specify what actions to take when a conflict occurs. Not all shift/reduce conflicts can be successfully resolved this way because the conflict may be due to something other than ambiguity, so incautious use of these facilities can cause the language accepted by the parser to be much different from that which was intended. The description file shall contain sufficient information to understand the cause of the conflict. Where ambiguity is the reason either the default or explicit rules should be adequate to produce a working parser.

The declared precedences and associativities (see **Declarations Section** (on page 3264)) are used to resolve parsing conflicts as follows:

1. A precedence and associativity is associated with each grammar rule; it is the precedence and associativity of the last token or literal in the body of the rule. If the **%prec** keyword is used, it overrides this default. Some grammar rules might not have both precedence and associativity.

2. If there is a shift/reduce conflict, and both the grammar rule and the input symbol have precedence and associativity associated with them, then the conflict is resolved in favor of the action (shift or reduce) associated with the higher precedence. If the precedences are the same, then the associativity is used; left associative implies reduce, right associative implies shift, and non-associative implies an error in the string being parsed.

3. When there is a shift/reduce conflict that cannot be resolved by rule 2, the shift is done. Conflicts resolved this way are counted in the diagnostic output described in **Error Handling**.

4. When there is a reduce/reduce conflict, a reduction is done by the grammar rule that occurs earlier in the input sequence. Conflicts resolved this way are counted in the diagnostic output described in **Error Handling**.

Conflicts resolved by precedence or associativity shall not be counted in the shift/reduce and reduce/reduce conflicts reported by *yacc* on either standard error or in the description file.

Error Handling

The token **error** shall be reserved for error handling. The name **error** can be used in grammar rules. It indicates places where the parser can recover from a syntax error. The default value of **error** shall be 256. Its value can be changed using a **%token** declaration. The lexical analyzer should not return the value of **error**.

The parser shall detect a syntax error when it is in a state where the action associated with the lookahead symbol is **error**. A semantic action can cause the parser to initiate error handling by executing the macro YYERROR. When YYERROR is executed, the semantic action passes control back to the parser. YYERROR cannot be used outside of semantic actions.

When the parser detects a syntax error, it normally calls *yyerror*() with the character string `"syntax error"` as its argument. The call shall not be made if the parser is still recovering

from a previous error when the error is detected. The parser is considered to be recovering from a previous error until the parser has shifted over at least three normal input symbols since the last error was detected or a semantic action has executed the macro *yyerrok*. The parser shall not call *yyerror*( ) when YYERROR is executed.

The macro function YYRECOVERING shall return 1 if a syntax error has been detected and the parser has not yet fully recovered from it. Otherwise, zero shall be returned.

When a syntax error is detected by the parser, the parser shall check if a previous syntax error has been detected. If a previous error was detected, and if no normal input symbols have been shifted since the preceding error was detected, the parser checks if the lookahead symbol is an endmarker (see **Interface to the Lexical Analyzer**). If it is, the parser shall return with a non-zero value. Otherwise, the lookahead symbol shall be discarded and normal parsing shall resume.

When YYERROR is executed or when the parser detects a syntax error and no previous error has been detected, or at least one normal input symbol has been shifted since the previous error was detected, the parser shall pop back one state at a time until the parse stack is empty or the current state allows a shift over **error**. If the parser empties the parse stack, it shall return with a non-zero value. Otherwise, it shall shift over **error** and then resume normal parsing. If the parser reads a lookahead symbol before the error was detected, that symbol shall still be the lookahead symbol when parsing is resumed.

The macro *yyerrok* in a semantic action shall cause the parser to act as if it has fully recovered from any previous errors. The macro *yyclearin* shall cause the parser to discard the current lookahead token. If the current lookahead token has not yet been read, *yyclearin* shall have no effect.

The macro YYACCEPT shall cause the parser to return with the value zero. The macro YYABORT shall cause the parser to return with a non-zero value.

Interface to the Lexical Analyzer

The *yylex*( ) function is an integer-valued function that returns a *token number* representing the kind of token read. If there is a value associated with the token returned by *yylex*( ) (see the discussion of *tag* above), it shall be assigned to the external variable *yylval*.

If the parser and *yylex*( ) do not agree on these token numbers, reliable communication between them cannot occur. For (single-byte character) literals, the token is simply the numeric value of the character in the current character set. The numbers for other tokens can either be chosen by *yacc*, or chosen by the user. In either case, the **#define** construct of C is used to allow *yylex*( ) to return these numbers symbolically. The **#define** statements are put into the code file, and the header file if that file is requested. The set of characters permitted by *yacc* in an identifier is larger than that permitted by C. Token names found to contain such characters shall not be included in the **#define** declarations.

If the token numbers are chosen by *yacc*, the tokens other than literals shall be assigned numbers greater than 256, although no order is implied. A token can be explicitly assigned a number by following its first appearance in the declarations section with a number. Names and literals not defined this way retain their default definition. All token numbers assigned by *yacc* shall be unique and distinct from the token numbers used for literals and user-assigned tokens. If duplicate token numbers cause conflicts in parser generation, *yacc* shall report an error; otherwise, it is unspecified whether the token assignment is accepted or an error is reported.

The end of the input is marked by a special token called the *endmarker*, which has a token number that is zero or negative. (These values are invalid for any other token.) All lexical analyzers shall return zero or negative as a token number upon reaching the end of their input. If

the tokens up to, but excluding, the endmarker form a structure that matches the start symbol, the parser shall accept the input. If the endmarker is seen in any other context, it shall be considered an error.

Completing the Program

In addition to *yyparse*( ) and *yylex*( ), the functions *yyerror*( ) and *main*( ) are required to make a complete program. The application can supply *main*( ) and *yyerror*( ), or those routines can be obtained from the *yacc* library.

Yacc Library

The following functions shall appear only in the *yacc* library accessible through the **–l y** operand to *c99*; they can therefore be redefined by a conforming application:

int *main*(**void**)
This function shall call *yyparse*( ) and exit with an unspecified value. Other actions within this function are unspecified.

int *yyerror*(**const char** **s*)
This function shall write the NUL-terminated argument to standard error, followed by a <newline>.

The order of the **–l y** and **–l l** operands given to *c99* is significant; the application shall either provide its own *main*( ) function or ensure that **–l y** precedes **–l l**.

Debugging the Parser

The parser generated by *yacc* shall have diagnostic facilities in it that can be optionally enabled at either compile time or at runtime (if enabled at compile time). The compilation of the runtime debugging code is under the control of YYDEBUG, a preprocessor symbol. If YYDEBUG has a non-zero value, the debugging code shall be included. If its value is zero, the code shall not be included.

In parsers where the debugging code has been included, the external **int** *yydebug* can be used to turn debugging on (with a non-zero value) and off (zero value) at runtime. The initial value of *yydebug* shall be zero.

When **–t** is specified, the code file shall be built such that, if YYDEBUG is not already defined at compilation time (using the *c99* **–D** YYDEBUG option, for example), YYDEBUG shall be set explicitly to 1. When **–t** is not specified, the code file shall be built such that, if YYDEBUG is not already defined, it shall be set explicitly to zero.

The format of the debugging output is unspecified but includes at least enough information to determine the shift and reduce actions, and the input symbols. It also provides information about error recovery.

Algorithms

The parser constructed by *yacc* implements an LALR(1) parsing algorithm as documented in the literature. It is unspecified whether the parser is table-driven or direct-coded.

A parser generated by *yacc* shall never request an input symbol from *yylex*( ) while in a state where the only actions other than the error action are reductions by a single rule.

The literature of parsing theory defines these concepts.

Limits

The *yacc* utility may have several internal tables. The minimum maximums for these tables are shown in the following table. The exact meaning of these values is implementation-defined. The implementation shall define the relationship between these values and between them and any error messages that the implementation may generate should it run out of space for any internal structure. An implementation may combine groups of these resources into a single pool as long as the total available to the user does not fall below the sum of the sizes specified by this section.

Table 4-22 Internal Limits in *yacc*

Limit	Minimum Maximum	Description
{NTERMS}	126	Number of tokens.
{NNONTERM}	200	Number of non-terminals.
{NPROD}	300	Number of rules.
{NSTATES}	600	Number of states.
{MEMSIZE}	5200	Length of rules. The total length, in names (tokens and non-terminals), of all the rules of the grammar. The left-hand side is counted for each rule, even if it is not explicitly repeated, as specified in **Grammar Rules in yacc** (on page 3266).
{ACTSIZE}	4000	Number of actions. ‘‘Actions’’ here (and in the description file) refer to parser actions (shift, reduce, and so on) not to semantic actions defined in **Grammar Rules in yacc** (on page 3266).

EXIT STATUS

The following exit values shall be returned:

0 Successful completion.

>0 An error occurred.

CONSEQUENCES OF ERRORS

If any errors are encountered, the run is aborted and *yacc* exits with a non-zero status. Partial code files and header files files may be produced. The summary information in the description file always shall be produced if the **–v** flag is present.

APPLICATION USAGE

Historical implementations experience name conflicts on the names **yacc.tmp**, **yacc.acts**, **yacc.debug**, **y.tab.c**, **y.tab.h**, and **y.output** if more than one copy of *yacc* is running in a single directory at one time. The **–b** option was added to overcome this problem. The related problem of allowing multiple *yacc* parsers to be placed in the same file was addressed by adding a **–p** option to override the previously hard-coded **yy** variable prefix.

The description of the **–p** option specifies the minimal set of function and variable names that cause conflict when multiple parsers are linked together. YYSTYPE does not need to be changed. Instead, the programmer can use **–b** to give the header files for different parsers different names, and then the file with the *yylex*( ) for a given parser can include the header for that parser. Names such as *yyclearerr* do not need to be changed because they are used only in the actions; they do not have linkage. It is possible that an implementation has other names, either internal ones for implementing things such as *yyclearerr*, or providing non-standard features that it wants to change with **–p**.

Unary operators that are the same token as a binary operator in general need their precedence adjusted. This is handled by the **%prec** advisory symbol associated with the particular grammar rule defining that unary operator. (See **Grammar Rules in yacc** (on page 3266).) Applications are not required to use this operator for unary operators, but the grammars that do not require it are rare.

EXAMPLES

Access to the *yacc* library is obtained with library search operands to *c99*. To use the *yacc* library *main*( ):

```
c99 y.tab.c –l y
```

Both the *lex* library and the *yacc* library contain *main*( ). To access the *yacc main*( ):

```
c99 y.tab.c lex.yy.c –l y –l l
```

This ensures that the *yacc* library is searched first, so that its *main*( ) is used.

The historical *yacc* libraries have contained two simple functions that are normally coded by the application programmer. These functions are similar to the following code:

```
#include <locale.h>
int main(void)
{
    extern int yyparse();
    setlocale(LC_ALL, "");
    /* If the following parser is one created by lex, the
       application must be careful to ensure that LC_CTYPE
       and LC_COLLATE are set to the POSIX locale. */
    (void) yyparse();
    return (0);
}
#include <stdio.h>
int yyerror(const char *msg)
{
    (void) fprintf(stderr, "%s\n", msg);
    return (0);
}
```

RATIONALE

The references in **Referenced Documents** (on page xx) may be helpful in constructing the parser generator. The referenced DeRemer and Pennello article (along with the works it references) describes a technique to generate parsers that conform to this volume of IEEE Std 1003.1-200x. Work in this area continues to be done, so implementors should consult current literature before doing any new implementations. The original Knuth article is the theoretical basis for this kind of parser, but the tables it generates are impractically large for reasonable grammars and should not be used. The ‘‘equivalent to’’ wording is intentional to assure that the best tables that are LALR(1) can be generated.

There has been confusion between the class of grammars, the algorithms needed to generate parsers, and the algorithms needed to parse the languages. They are all reasonably orthogonal. In particular, a parser generator that accepts the full range of LR(1) grammars need not generate a table any more complex than one that accepts SLR(1) (a relatively weak class of LR grammars) for a grammar that happens to be SLR(1). Such an implementation need not recognize the case, either; table compression can yield the SLR(1) table (or one even smaller than that) without

recognizing that the grammar is SLR(1). The speed of an LR(1) parser for any class is dependent more upon the table representation and compression (or the code generation if a direct parser is generated) than upon the class of grammar that the table generator handles.

The speed of the parser generator is somewhat dependent upon the class of grammar it handles. However, the original Knuth article algorithms for constructing LR parsers was judged by its author to be impractically slow at that time. Although full LR is more complex than LALR(1), as computer speeds and algorithms improve, the difference (in terms of acceptable wall-clock execution time) is becoming less significant.

Potential authors are cautioned that the referenced DeRemer and Pennello article previously cited identifies a bug (an over-simplification of the computation of LALR(1) lookahead sets) in some of the LALR(1) algorithm statements that preceded it to publication. They should take the time to seek out that paper, as well as current relevant work, particularly Aho's.

The **–b** option was added to provide a portable method for permitting *yacc* to work on multiple separate parsers in the same directory. If a directory contains more than one *yacc* grammar, and both grammars are constructed at the same time (by, for example, a parallel *make* program), conflict results. While the solution is not historical practice, it corrects a known deficiency in historical implementations. Corresponding changes were made to all sections that referenced the filenames **y.tab.c** (now ''the code file''), **y.tab.h** (now ''the header file''), and **y.output** (now ''the description file'').

The grammar for *yacc* input is based on System V documentation. The textual description shows there that the `';'` is required at the end of the rule. The grammar and the implementation do not require this. (The use of **C_IDENTIFIER** causes a reduce to occur in the right place.)

Also, in that implementation, the constructs such as **%token** can be terminated by a semicolon, but this is not permitted by the grammar. The keywords such as **%token** can also appear in uppercase, which is again not discussed. In most places where `'%'` is used, `'\'` can be substituted, and there are alternate spellings for some of the symbols (for example, **%LEFT** can be "%<" or even "\<").

Historically, <*tag*> can contain any characters except `'>'`, including white space, in the implementation. However, since the *tag* must reference a ISO C standard union member, in practice conforming implementations need to support only the set of characters for ISO C standard identifiers in this context.

Some historical implementations are known to accept actions that are terminated by a period. Historical implementations often allow `'$'` in names. A conforming implementation does not need to support either of these behaviors.

Deciding when to use **%prec** illustrates the difficulty in specifying the behavior of *yacc*. There may be situations in which the *grammar* is not, strictly speaking, in error, and yet *yacc* cannot interpret it unambiguously. The resolution of ambiguities in the grammar can in many instances be resolved by providing additional information, such as using **%type** or **%union** declarations. It is often easier and it usually yields a smaller parser to take this alternative when it is appropriate.

The size and execution time of a program produced without the runtime debugging code is usually smaller and slightly faster in historical implementations.

Statistics messages from several historical implementations include the following types of information:

```
n/512 terminals, n/300 non-terminals
n/600 grammar rules, n/1500 states
n shift/reduce, n reduce/reduce conflicts reported
```

```
n/350 working sets used
Memory: states, etc. n/15000, parser n/15000
n/600 distinct lookahead sets
n extra closures
n shift entries, n exceptions
n goto entries
n entries saved by goto default
Optimizer space used: input n/15000, output n/15000
n table entries, n zero
Maximum spread: n, Maximum offset: n
```

The report of internal tables in the description file is left implementation-defined because all aspects of these limits are also implementation-defined. Some implementations may use dynamic allocation techniques and have no specific limit values to report.

The format of the **y.output** file is not given because specification of the format was not seen to enhance applications portability. The listing is primarily intended to help human users understand and debug the parser; use of **y.output** by a conforming application script would be unusual. Furthermore, implementations have not produced consistent output and no popular format was apparent. The format selected by the implementation should be human-readable, in addition to the requirement that it be a text file.

Standard error reports are not specifically described because they are seldom of use to conforming applications and there was no reason to restrict implementations.

Some implementations recognize `"={"` as equivalent to `'{'` because it appears in historical documentation. This construction was recognized and documented as obsolete as long ago as 1978, in the referenced *Yacc: Yet Another Compiler-Compiler*. This volume of IEEE Std 1003.1-200x chose to leave it as obsolete and omit it.

Multi-byte characters should be recognized by the lexical analyzer and returned as tokens. They should not be returned as multi-byte character literals. The token **error** that is used for error recovery is normally assigned the value 256 in the historical implementation. Thus, the token value 256, which used in many multi-byte character sets, is not available for use as the value of a user-defined token.

FUTURE DIRECTIONS

None.

SEE ALSO

c99, *lex*

CHANGE HISTORY

First released in Issue 2.

Issue 5

FUTURE DIRECTIONS section added.

Issue 6

This utility is now marked as part of the C-Language Development Utilities option.

Minor changes have been added to align with the IEEE P1003.2b draft standard.

The normative text is reworded to avoid use of the term ‘‘must’’ for application requirements.

IEEE PASC Interpretation 1003.2 #177 is applied, changing the comment on **RCURL** from the **}%** token to the **%}**.

41331 **NAME**

41332 zcat — expand and concatenate data

41333 **SYNOPSIS**

41334 XSI
```
zcat [file...]
```
41335

41336 **DESCRIPTION**

41337 The *zcat* utility shall write to standard output the uncompressed form of files that have been
41338 compressed using the *compress* utility. It is the equivalent of *uncompress* –**c**. Input files are not
41339 affected.

41340 **OPTIONS**

41341 None.

41342 **OPERANDS**

41343 The following operand shall be supported:

41344 *file* The pathname of a file previously processed by the *compress* utility. If *file* already
41345 has the **.Z** suffix specified, it is used as submitted. Otherwise, the **.Z** suffix is
41346 appended to the filename prior to processing.

41347 **STDIN**

41348 The standard input shall be used only if no *file* operands are specified, or if a *file* operand is `'–'`.

41349 **INPUT FILES**

41350 Input files shall be compressed files that are in the format produced by the *compress* utility.

41351 **ENVIRONMENT VARIABLES**

41352 The following environment variables shall affect the execution of *zcat*:

41353 *LANG* Provide a default value for the internationalization variables that are unset or null.
41354 (See the Base Definitions volume of IEEE Std 1003.1-200x, Section 8.2,
41355 Internationalization Variables for the precedence of internationalization variables
41356 used to determine the values of locale categories.)

41357 *LC_ALL* If set to a non-empty string value, override the values of all the other
41358 internationalization variables.

41359 *LC_CTYPE* Determine the locale for the interpretation of sequences of bytes of text data as
41360 characters (for example, single-byte as opposed to multi-byte characters in
41361 arguments).

41362 *LC_MESSAGES*
41363 Determine the locale that should be used to affect the format and contents of
41364 diagnostic messages written to standard error.

41365 *NLSPATH* Determine the location of message catalogs for the processing of *LC_MESSAGES*.

41366 **ASYNCHRONOUS EVENTS**

41367 Default.

41368 **STDOUT**

41369 The compressed files given as input shall be written on standard output in their uncompressed
41370 form.

41371 **STDERR**

41372 The standard error shall be used only for diagnostic messages.

41373 **OUTPUT FILES**

41374 None.

41375 **EXTENDED DESCRIPTION**

41376 None.

41377 **EXIT STATUS**

41378 The following exit values shall be returned:

41379 0 Successful completion.

41380 >0 An error occurred.

41381 **CONSEQUENCES OF ERRORS**

41382 Default.

41383 **APPLICATION USAGE**

41384 None.

41385 **EXAMPLES**

41386 None.

41387 **RATIONALE**

41388 None.

41389 **FUTURE DIRECTIONS**

41390 None.

41391 **SEE ALSO**

41392 *compress*, *uncompress*

41393 **CHANGE HISTORY**

41394 First released in Issue 4.