-- --

IEEE P1003.2 Draft 11.2 – September 1991

4.34.3 Options

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

The following options shall be supported by the implementation:

`−a` Write information 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 4.34.6.1.

`−k` Write the names and values of selected keywords. The implementation may omit values for some keywords; see 4.34.4.

`−m` Write names of available charmaps; see 2.4.1. 1

4.34.4 Operands

The following operand shall be supported by the implementation:

name The name of a locale category as defined in 2.5, 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.

4.34.5 External Influences

4.34.5.1 Standard Input

None.

4.34.5.2 Input Files

None.

4.34.5.3 Environment Variables

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

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

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

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

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

The **LANG** and **LC_*** environment variables shall specify the current locale environment to be written out; they shall be used if the `-a` option is not specified.

4.34.5.4 Asynchronous Events

Default.

4.34.6 External Effects

4.34.6.1 Standard Output

If `locale` is invoked without any options or operands, the names and values of the **LANG** and **LC_*** environment variables described in this standard shall be written to the standard output, one variable per line, with **LANG** first, and each line using the following format. Only those variables set in the environment and not overridden by **LC_ALL** shall be written using this format:

`"%s=%s\n"`, *<variable_name>* , *<value>*

The names of those **LC_*** variables associated with locale categories defined in this standard that are not set in the environment or are overridden by **LC_ALL** shall be written in the following format:

`"%s=\"%s\"\n"`, *<variable_name>* , *<implied value>*

The *<implied value>* shall be the name of the locale that has been selected for that category by the implementation, based on the values in **LANG** and **LC_ALL**, as described in 2.6.

The *<value>* and *<implied value>* shown above shall be properly quoted for possi- 1
ble later re-entry to the shell. The *<value>* shall not be quoted using double- 1
quotes (so that it can be distinguished by the user from the *<implied value>* case, 1
which always requires double-quotes). 1

The **LC_ALL** variable shall be written last, using the first format shown above. If it is not set, it shall be written as: 1

`"LC_ALL=\n"`

If any arguments are specified:

(1) If the `-a` option is specified, the names of all the public locales shall be written, each in the following format:

`"%s\n"`, *<locale name>*

(2) If the `-c` option is specified, the name(s) of all selected categories shall be written, each in the following format:

`"%s\n"`, *<category name>*

If keywords are also selected for writing (see following items), the category name output shall precede the keyword output for that category.

If the `-c` option is not specified, the names of the categories shall not be written; only the keywords, as selected by the *name* operand, shall be written. 2

(3) If the `-k` option is specified, the name(s) and value(s) of selected keywords shall be written. If a value is nonnumeric, it shall be written in the following format:

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

If the keyword was `charmap`, the name of the charmap (if any) that was specified via the `localedef -f` option when the locale was created shall be written, with the word `charmap` as *<keyword name>* .

If a value is numeric, it shall be written in one of the following formats:

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

`"%s=%c%o\n"`, *<keyword name>* , *<escape character>* , *<keyword value>*

`"%s=%cx%x\n"`, *<keyword name>* , *<escape character>* , *<keyword value>*

where the *<escape character>* is that identified by the `escape_char` keyword in the current locale; see 2.5.2.

Compound keyword values (list entries) shall be separated in the output by semicolons. When included in keyword values, the semicolon, the double-quote, the backslash, and any control character shall be preceded (escaped) with the escape character.

(4) If the `-k` option is not specified, selected keyword values shall be written, each in the following format:

`"%s\n"`, *<keyword value>*

If the keyword was `charmap`, the name of the charmap (if any) that was specified via the `localedef −f` option when the locale was created shall be written.

(5) If the `−m` option is specified, then a list of all available charmaps shall be written, each in the format

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

where *<charmap>* is in a format suitable for use as the option-argument to the `localedef −f` option.

4.34.6.2 Standard Error

Used only for diagnostic messages.

4.34.6.3 Output Files

None.

4.34.7 Extended Description

None.

4.34.8 Exit Status

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

0 All the requested information was found and output successfully.

>0 An error occurred.

4.34.9 Consequences of Errors

Default.

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

Examples, Usage

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                                                    1
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 standard.

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:

```
if printf "%s\n" "$response" | grep -Eq "$(locale yesexpr)"
then
        affirmative processing goes here
else
        nonaffirmative processing goes here
fi
```

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 2.6.

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, e.g., which character classes are available).

History of Decisions Made

This command was added in Draft 9 to resolve objections to the lack of a way for applications to determine what locales are available, a way to examine the contents of existing public locales, a way to retrieve specific locale items, and a way to recognize affirmative and negative responses in an international environment.

In Draft 10 it was cut back considerably in answer to balloting objections about its complexity and requirement of features not useful for application programs. The format for the no-arguments case was expanded to show the implied values of the categories as an aid to the novice user; the output was of little more value than that from `env`.

Based on the questionable value in a shell script of getting an entire array of characters back, and the problem of returning a collation description that makes sense, short of a complete `localedef` source, the output from requests for categories LC_CTYPE and LC_COLLATE has been made implementation defined.

The `-m` option has been added to allow applications to query for the existence of charmaps. The output is a list of the charmaps (implementation-supplied and user-supplied, if any) on the system.

The `-c` option was included for readability when more than one category is 2
selected (e.g., via more than one keyword name or via a category name). It is 2
valid both with and without the `-k` option. 2

The `charmap` keyword, which returns the name of the charmap (if any) that was used when the current locale was created, was introduced to allow applications needing the information to retrieve it.

4.35 `localedef` — Define locale environment

4.35.1 Synopsis

`localedef [-c][-f` *charmap* `][-i` *sourcefile* `]` *name*

4.35.2 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 2.5. It is implementation defined whether users shall have the capability to create new locales, in addition to those supplied by the implementation. If the symbolic constant {POSIX 2_LOCALEDEF } is defined, then the system supports the creation of new locales. In a system not supporting this capability, the `localedef` utility shall terminate with an exit code of 3.

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 nonmonetary 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.

4.35.3 Options

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

The following options shall be supported by the implementation:

`−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 2.4.1. This option shall be 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 default character mapping file shall be used. 2

`−i` *inputfile* The pathname of a file containing the source definitions. If this option is not present, source definitions shall be read from standard input. The format of the *inputfile* is described in 2.5.2.

4.35.4 Operands

The following operand shall be supported by the implementation:

name Identifies the locale. See 2.5 for a description of the use of this name. If the name contains one or more slash characters, *name* shall be interpreted as a pathname where the created locale definition(s) shall be stored. If *name* does not contain any slash characters, the interpretation of the name is implementation defined and the locale shall be public. This capability may be restricted to users with appropriate privileges.

4.35.5 External Influences

4.35.5.1 Standard Input

Unless the `-i` option is specified, the standard input shall be a text file containing
one or more locale category source definitions, as described in 2.5.2. When lines 1
are continued using the escape character mechanism, there is no limit to the 1
length of the accumulated continued line. 1

4.35.5.2 Input Files

The character set mapping file specified as the *charmap* option-argument is described under 2.4.1. If a locale category source definition contains a `copy` statement, as defined in 2.5.2, and the `copy` statement names a valid, existing locale, then `localedef` shall behave as if the source definition had contained a valid category source definition for the named locale.

4.35.5.3 Environment Variables

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

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

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**. and **LC_*** variables as described in 2.6.

LC_COLLATE (This variable shall have no affect on `localedef`; the POSIX Locale shall be used for this category.)

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of argument data as characters (e.g., single- versus multibyte characters). This variable shall have no affect on the processing of `localedef` input data; the POSIX Locale shall be used for this purpose, regardless of the value of this variable.

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

4.35.5.4 Asynchronous Events

Default.

4.35.6 External Effects

4.35.6.1 Standard Output

The utility shall report all categories successfully processed, in an unspecified format.

4.35.6.2 Standard Error

Used only for diagnostic messages.

4.35.6.3 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.

4.35.7 Extended Description

None.

4.35.8 Exit Status

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

0 No errors occurred and the locale(s) were successfully created.

1 Warnings occurred and the locale(s) 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.

4.35.9 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 conditions).

— 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 1
the source. 1

Other implementation-defined conditions may also cause warnings.

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

Usage, Examples

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.)

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 code set). To aid portability, all *charmap* definitions shall use the same symbolic names for the portable character set. As explained in 2.4.1, 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-provided *charmap* is named.

History of Decisions Made

This description is based on work performed in the UniForum Technical Committee Subcommittee on Internationalization.

The `localedef` utility is provided as a standard, portable interface for implementations that allow users to create new locales, in addition to implementation-supplied ones.

The ability to create new locales and categories, already available on many commercially available implementations of POSIX compliant systems, provides the means by which application providers can develop portable applications which use standard interfaces to adjust the behavior of the application to language and culture differences.

4.36 `logger` — Log messages

4.36.1 Synopsis

`logger` *string* ...

4.36.2 Description

The `logger` utility saves a message, in an unspecified manner and format, containing the *string* operands provided by the user. The messages are expected to be evaluated later by personnel performing system administration tasks.

4.36.3 Options

None.

4.36.4 Operands

The following operands shall be supported by the implementation:

string One of the string arguments whose contents are concatenated together, in the order specified, separated by single `<space>`s.

4.36.5 External Influences

4.36.5.1 Standard Input

None.

4.36.5.2 Input Files

None.

4.36.5.3 Environment Variables

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

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

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

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

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

4.36.5.4 Asynchronous Events

Default.

4.36.6 External Effects

4.36.6.1 Standard Output

None.

4.36.6.2 Standard Error

Used only for diagnostic messages.

4.36.6.3 Output Files

Unspecified.

4.36.7 Extended Description

None.

4.36.8 Exit Status

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

0 Successful completion.

>0 An error occurred.

4.36.9 Consequences of Errors

Default.

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

Examples, Usage

This utility allows logging of information for later use by a system administrator or programmer in determining why noninteractive utilities have failed. POSIX.2 makes no requirements for the locations of the saved message, their format, or retention period. It also provides no method for a portable application to read messages, once written. (It is expected that the POSIX.7 System Administration standard will have something to say about that.)

The purpose of this utility might best be illustrated by an example. A batch application, running noninteractively, tries to read a configuration file and fails; it may attempt to notify the system administrator with:

```
logger myname: unable to read file foo. [time stamp]
```

The text with **LC_MESSAGES** about diagnostic messages means diagnostics from `logger` to the user or application, not diagnostic messages that the user is sending to the system administrator.

History of Decisions Made

Multiple *string* arguments were allowed, similar to `echo`, for ease of use.

In Draft 9, the `posixlog` utility was renamed `logger` to match its BSD forebear, with which it is (downward) compatible.

The working group believed strongly that some method of alerting administrators to errors was necessary. The obvious example is a batch utility, running noninteractively, that is unable to read its configuration files, or that is unable to create or write its results file. However, the working group did not wish to define the format or delivery mechanisms as they have historically been (and will probably continue to be) very system specific, as well as involving functionality clearly outside of the scope of this standard.

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

4.37 `logname` — Return user's login name

4.37.1 Synopsis

```
logname
```

4.37.2 Description

The `logname` utility shall write the user's login name to standard output. The login name shall be the string that would be returned by the POSIX.1 {8} *getlogin*() function. Under the conditions where the *getlogin*() function would fail, the `logname` utility shall write a diagnostic message to standard error and exit with a nonzero exit status.

4.37.3 Options

None.

4.37.4 Operands

None.

4.37.5 External Influences

4.37.5.1 Standard Input

None.

4.37.5.2 Input Files

None.

4.37.5.3 Environment Variables

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

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

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

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

4.37.5.4 Asynchronous Events

Default.

4.37.6 External Effects

4.37.6.1 Standard Output

The `logname` utility output shall be a single line consisting of the user's login name:

`"%s\n"`, *<login name>*

4.37.6.2 Standard Error

Used only for diagnostic messages.

4.37.6.3 Output Files

None.

4.37.7 Extended Description

None.

4.37.8 Exit Status

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

0 Successful completion.

>0 An error occurred.

4.37.9 Consequences of Errors

Default.

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

Examples, Usage

The `logname` utility explicitly ignores the **LOGNAME** environment variable because environment changes could produce erroneous results.

History of Decisions Made

The `passwd` file is not listed as required, because the implementation may have other means of mapping login names.

4.38 `lp` — Send files to a printer

4.38.1 Synopsis

`lp [−c] [−d` *dest* `] [−n` *copies* `] [`*file* ... `]`

4.38.2 Description

The `lp` utility shall copy the input files to an output device in an unspecified manner. The default output destination should be to a hardcopy device, such as a printer or microfilm recorder, that produces nonvolatile, human-readable documents. If such a device is not available to the application, or if the system provides no such device, the `lp` utility shall exit with a nonzero exit status.

The actual writing to the output device may occur some time after the `lp` utility successfully exits. During the portion of the writing that corresponds to each input file, the implementation shall guarantee exclusive access to the device.

4.38.3 Options

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

The following options shall be supported by the implementation:

`−c` Exit only after further access to any of the input files is no longer required. The application can then safely delete or modify the files without affecting the output operation.

`−d` *dest* Specify a string that names the output device or destination. If `−d` is not specified, and neither the **LPDEST** nor **PRINTER** environment variable is set, an unspecified output device is used. The `−d` *dest* option shall take precedence over **LPDEST** , which in turn shall take precedence over **PRINTER** . Results are undefined when *dest* contains a value that is not a valid device or destination name.

−n *copies* Write *copies* number of copies of the files, where *copies* is a positive decimal integer. The methods for producing multiple copies and for arranging the multiple copies when multiple *file* operands are used are unspecified, except that each file shall be output as an integral whole, not interleaved with portions of other files.

4.38.4 Operands

The following operands shall be supported by the implementation:

file A pathname of a file to be output. If no *file* operands are specified, or if a *file* operand is −, the standard input shall be used. If a *file* operand is used, but the −c option is not specified, the process performing the writing to the output device may have user and group permissions that differ from that of the process invoking lp.

4.38.5 External Influences

4.38.5.1 Standard Input

The standard input shall be used only if no *file* operands are specified, or if a *file* operand is −. See Input Files.

4.38.5.2 Input Files

The input files shall be text files.

4.38.5.3 Environment Variables

The following environment variables shall affect the execution of lp:

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

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

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

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

LPDEST This variable shall be interpreted as a string that names the output device or destination. If the **LPDEST** environment variable is not set, the **PRINTER** environment variable shall be used. The −d *dest* option shall take precedence over **LPDEST** . Results are undefined when −d is not specified and **LPDEST** contains a value that is not a valid device or destination name.

PRINTER This variable shall be interpreted as a string that names the output device or destination. If the **LPDEST** and **PRINTER** environment variables are not set, an unspecified output device is used. The −d *dest* option and the **LPDEST** environment variable shall take precedence over **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.

4.38.5.4 Asynchronous Events

Default.

4.38.6 External Effects

4.38.6.1 Standard Output

A message concerning the identification or status of the print request may be 2
written, in an unspecified format. 2

4.38.6.2 Standard Error

Used only for diagnostic messages.

4.38.6.3 Output Files

None.

4.38.7 Extended Description

None.

4.38.8 Exit Status

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

0 All input files were processed successfully.

>0 No output device was available, or an error occurred.

4.38.9 Consequences of Errors

Default.

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

Examples, Usage

Since the default destination, device type, queueing mechanisms, and acceptable forms of input are all unspecified, usage guidelines for what a portable application can do are as follows:

(1) 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.

(2) 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 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 implementation's default page size.

Alternatively, the application can arrange its installation in such a way that 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 the standard tells the industry that portable 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 clearly shows the working group's intent, even if it cannot mandate that all systems (such as laptops) have printers.

Examples :

To print file *file*:

```
lp −c file
```

To print multiple files with headers:

```
pr file1 file2 | lp
```

On most existing implementations of `lp`, an option is provided to pass printer specific options to the daemon handling the printer. It is not specified here because the printer-specific options are widespread and in conflict, the `lp` specified here is not required to even have a queueing mechanism, and the choice of options varies widely from printer to printer. Nonetheless, implementors are encouraged to use this mechanism where appropriate:

`−o` *option* Specifies an implementation-defined option that controls the specific operation of the printer. The following *option* s could be used for the meanings below if the hardware is capable of supporting the option.

option	**Meaning**
`lp2`	two logical pages per physical page
`lp4`	four logical pages per physical page
`d`	double sided

POSIX. 2 does not specify what the ownership of the process performing the writ- 1
ing to the output device may be. If `−c` is not used, it is unspecified whether the 1
process performing the writing to the output device will have permission to read 1
file if there are any restrictions in place on who may read *file* until after it is 1
printed. Also, if `−c` is not used, the results of deleting *file* before it is printed are 1
unspecified. 1

History of Decisions Made

The `lp` utility was designed to be a basic version of a utility that is already available in many historical implementations. The working group felt 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 working group 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.

The standard makes no representations concerning the format of the printed output, except that it must be ‘‘human-readable’’ and ‘‘nonvolatile.’’ 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.)

A portable application will 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.

The standard 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 jumbled up with another, but there is no statement that all the files or copies have to