-- --

IEEE P1003.2 Draft 11.2 – September 1991

that BSD has used 512-byte units in some places and 1024-byte units in other places. (System V has consistently used 512.) Therefore, POSIX.2 and POSIX.2a usually specify 512 and that value has been restored here as it was in Draft 9. Future releases of BSD are expected to consistently provide 512 as a default with a way of specifying 1024-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 standard, as the appropriate vehicle is a messaging system; i.e., the format should be specified as a ‘‘message.’’

4.40 `mailx` — Process messages

4.40.1 Synopsis

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

4.40.2 Description

The `mailx` utility shall read standard input and send it to one or more addresses in an unspecified manner. 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.

4.40.3 Options

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

The following option shall be supported by the implementation:

`−s` *subject* A string representing the subject of the message. All characters 2
in the *subject* string shall appear in the delivered message. The 2
results are unspecified if *subject* is longer than {LINE_MAX} − 10 2
bytes or contains a `<newline>`. 2

4.40.4 Operands

The following operand shall be supported by the implementation:

address Send a message to *address*. Valid login names on the local system shall be accepted as valid *address*es. The interpretation of other types of *address*es is unspecified. An implementation-defined way for a user with a login-name address to retrieve the message shall be provided by the implementation.

4.40.5 External Influences

4.40.5.1 Standard Input

The standard input shall be a text file. The results are unspecified if the first character of any input line is a tilde (`~`).

4.40.5.2 Input Files

None.

4.40.5.3 Environment Variables

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

DEAD This variable shall affect the processing of signals by `mailx`: if the application sets this variable to `/dev/null`, the results of receiving a signal are as described by this standard; they are otherwise unspecified.

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

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

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

LC_CTYPE This variable shall determine the 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.

MAILRC This variable shall affect the startup processing of `mailx`: if the application sets this variable to `/dev/null`, `mailx` shall operate as described by this standard; otherwise, unspecified results occur.

4.40.5.4 Asynchronous Events

Default.

4.40.6 External Effects

4.40.6.1 Standard Output

None.

4.40.6.2 Standard Error

Used only for diagnostic messages.

4.40.6.3 Output Files

None.

4.40.7 Extended Description

None.

4.40.8 Exit Status

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

- 0 Successful completion.
- >0 An error occurred.

4.40.9 Consequences of Errors

Default.

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

Usage, Examples

The intent is that a header indicating who sent the message and a message subject string, the contents of the standard input, and perhaps a trailer is delivered to users specified by the given addresses. The standard input, however, may have to be manipulated slightly to avoid confusion between message text and headers as it passes through the message delivery system. POSIX.2 does not specify how standard input may be manipulated; that will be specified in detail by POSIX.2a.

The restriction on a subject line being {LINE_MAX} − 10 bytes is based on the his- 2
torical format that consumes 10 bytes for `"Subject: "` and the trailing `<new-` 2
`line>`. Many historical mailers that a message may encounter on other systems 2
will not be able to handle lines that long, however. 2

History of Decisions Made

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

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

Implementations are encouraged to conform to the various delivery mechanisms described in ARPANET Requests for Comment Numbers 819, 822, 882, 920, 921, and the CCITT X.400 standards.

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

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

Before Draft 7, there was a utility named `mailto`. In Draft 7, the name was changed to `sendto` because of comments noting that `mailto` implied full mail-like functionality and that was not what the specification provided. However, there have been consistent comments that it does not make sense to end up with a standard that will require two mail-sending interfaces. (POSIX.2a is working on a fully fleshed-out mail-sending and -reading utility based on the historical System V `mailx` utility.) A message- (or mail-) sending utility that is a subset of the interactive utility that will be described by POSIX.2a is much more consistent with the rest of the standard. Therefore, in Draft 10 the name has been changed again to `mailx` and the description is a small subset of the functionality being specified by POSIX.2a. It provides a portable way for a shell script to be able to send a message to a user on the local system. It is expected that implementations that have provided `mailx` in the past will use it to meet the POSIX.2 requirements. Implementations that have not provided `mailx` in the past will be able to create a simple interface to their current mailer to meet these requirements.

Most of the features provided by `mailx` (and the similar BSD `Mail`) utility are not specified here because they are not needed for noninteractive use (applications do not usually read mail without user participation) and they depend on other interactive features that are not defined by POSIX.2, but will be defined by POSIX.2a (the `~v` command, for instance, uses the `vi` editor as a default.)

If the **DEAD** environment variable is not set to `/dev/null`, historical versions of `mailx` and `Mail` save a message being constructed in a file under some circumstances when some asynchronous events occur. The details will be specified by POSIX.2a.

If the **MAILRC** environment variable does not name an empty file, 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, applications are advised to set **MAILRC** to `/dev/null`. POSIX.2a will specify details on the format of the initialization file.

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.

4.41 `mkdir` — Make directories

4.41.1 Synopsis

`mkdir [−p][−m` *mode* `]` *dir* ...

4.41.2 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 POSIX.1 {8} *mkdir*() function, 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, 1
the *mode* option-argument overrides this default.) 1