IEEE P1003.2a Draft 8 – December 1991

5.28 `talk` — Talk to another user

5.28.1 Synopsis

`talk` *address* [*terminal*]

5.28.2 Description

The `talk` utility is a two-way, screen-oriented communication program.

When first invoked, `talk` shall send a message similar to:

`Message from` <*unspecified string*>
`talk: connection requested by` *your_address*
`talk: respond with: talk` *your_address*

to the specified *address* . At this point, the recipient of the message can reply by typing:

`talk` *your_address*

Once communication is established, the two parties can type simultaneously, with
their output displayed in separate regions of the screen. Characters shall be pro- 8
cessed as follows: 8

— Typing the `<alert>` character shall alert the recipient's terminal.

— Typing `control-L` shall cause the sender's screen regions to be refreshed.

— Typing the erase and kill characters shall affect the sender's terminal in the manner described by the POSIX.1 {8} *termios* interface.

— Typing the interrupt or end-of-file characters shall terminate the local `talk` utility. Once the `talk` session has been terminated on one side, the other side of the `talk` session shall be notified that the `talk` session has been terminated and shall be able to do nothing except exit.

— Typing characters from LC_CTYPE classifications `print` or `space` shall 8
cause those characters to be sent to the recipient's terminal.

— When and only when the `stty` *iexten* local mode (see 4.59.4.4) is enabled, 8
additional special control characters and multibyte or single-byte functions 8
shall be processed in an implementation-defined manner. (See POSIX.1 {8} 8
section 7.1.1.9.) 8

— Typing other nonprintable characters shall cause implementation-defined sequences of printable characters to be sent to the recipient's terminal.

Permission to be a recipient of a `talk` message can be denied or granted by use of
the `mesg` utility (see 5.17). However, a user's privilege may further constrain the 7
domain of accessibility of other users' terminals. The `talk` utility shall fail when 7
the user lacks the appropriate privileges to perform the requested action. 7

Certain block-mode terminals do not have all the capabilities necessary to support the simultaneous exchange of messages required for `talk`. When this type of exchange cannot be supported on such terminals, the implementation may support an exchange with reduced levels of simultaneous interaction or it may report an error describing the terminal-related deficiency.

5.28.3 Options

None.

5.28.4 Operands

The following operands shall be supported by the implementation:

address The recipient of the `talk` session. One form of *address* is the *<user name>*, as returned by the `who` utility (see 5.36). Other address formats and how they are handled are unspecified.

terminal If the recipient is logged in more than once, the *terminal* argument can be used to indicate the appropriate terminal name. If *terminal* is not specified, the `talk` message shall be displayed on one or more accessible terminals in use by the recipient. The format of *terminal* shall be the same as that returned by the `who` utility.

5.28.5 External Influences

5.28.5.1 Standard Input

Characters read from standard input shall be copied to the recipient's terminal in an unspecified manner. If standard input is not a terminal, talk shall write a diagnostic message and exit with a nonzero status.

5.28.5.2 Input Files

None.

5.28.5.3 Environment Variables

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

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

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

LC_CTYPE This variable shall determine the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files). If the recipient's locale does not use an LC_CTYPE equivalent to the sender's, the results are undefined.

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

TERM This variable shall be interpreted as the name of the
invoker's terminal type. If this variable is unset or null, 7
an unspecified default terminal type shall be used. 7

5.28.5.4 Asynchronous Events

When the `talk` utility receives a SIGINT signal, the utility shall terminate and exit with a zero status. It shall take the standard action for all other signals; see 2.11.5.4.

5.28.6 External Effects

5.28.6.1 Standard Output

If standard output is a terminal, characters copied from the recipient's standard input may be written to standard output. Standard output also may be used for diagnostic messages. If standard output is not a terminal, `talk` shall exit with a nonzero status.

5.28.6.2 Standard Error

None.

5.28.6.3 Output Files

None.

5.28.7 Extended Description

None.

5.28.8 Exit Status

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

- 0 Successful completion.
- >0 An error occurred, or `talk` was invoked on a terminal incapable of supporting it.

5.28.9 Consequences of Errors

Default.

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

Usage, Examples

The handling of nonprintable characters is partially implementation defined because the details of mapping them to printable sequences is not needed by the user. Historical implementations, for security reasons, disallow the transmission
of nonprintable characters that may send commands to the other terminal. 8
Because the handling of nonprintable, nonspace characters is tied to the `stty` 8
description of *iexten*, implementation extensions within the terminal driver can be 8
accessed. For example, some implementations provide line editing functions with 8
certain control character sequences. 8

History of Decisions Made

The `write` utility was included in the standard since it can be implemented on all terminal types. The `talk` utility, which cannot be implemented on certain terminals, was considered to be a ‘‘better’’ communications interface. Both of these programs are in widespread use on historical implementations. Therefore, both utilities have been specified.

All references to networking abilities (`talk`ing to a user on another system) were removed as being outside the scope of the document.

The version of `talk` released with 4.3 BSD uses a protocol that is incompatible with the protocol used in the version released with 4.2 BSD.

Historical BSD and System V versions of `talk` terminate both of the conversations when either user breaks out of the session. This can lead to adverse consequences if a user unwittingly continues to enter text that will be interpreted by the shell when the other terminates the session. Therefore, the version of `talk` specified by this standard requires both users to explicitly terminate their end of the session.

Only messages sent to the user’s own terminal can be internationalized in any way.

— The original ''Message from <*unspecified string*> ...'' message sent to the recipient's terminal cannot be internationalized because the recipient's environment is as yet inaccessible to the `talk` utility. The invoking party's environment is irrelevant.

— Subsequent communication between the two parties cannot be internation-
alized because the two parties may specify different languages in their
environment (and nonportable characters cannot be mapped from one 7
language to another).

— Neither party can be required to communicate in a language other than C and/or the one specified by their environment because unavailable terminal hardware support (e.g., fonts) may be required.

The text in Standard Output reflects the usage of the verb ''display'' in this clause; some `talk` implementations actually use standard output to write to the terminal, but POSIX.2 does not require that to be the case.

The format of the terminal name is unspecified, but the descriptions of `ps`, `talk`, 7
`who`, and `write` require that they all use or accept the same format. 7

5.29 `time` — Time a simple command

5.29.1 Synopsis

`time [−p]` *utility* `[`*argument* ...`]`

5.29.2 Description

The `time` utility shall invoke the utility named by the *utility* operand with arguments supplied as the *argument* operands and write a message to standard error that lists timing statistics for the utility. The message shall include the following information:

— The elapsed (''real'') time between invocation of *utility* and its termination.

— The User CPU time, equivalent to the sum of the *tms_utime* and *tms_cutime* fields returned by the POSIX.1 {8} *times*() function for the process in which *utility* is executed.

— The System CPU time, equivalent to the sum of the *tms_stime* and *tms_cstime* fields returned by the POSIX.1 {8} *times*() function for the process in which *utility* is executed.

The precision of the timing shall be no less than the granularity defined for the size of the clock tick unit on the system, but the results shall be reported in terms of standard time units (e.g., 0.02 seconds, 00:00:00.02, 1m33.75s, 365.21 seconds), not numbers of clock ticks.

When `time` is used as part of a pipeline, the times reported are unspecified, 8
except when it is the sole command within a grouping command (see 3.9.4.1) in 8
that pipeline. For example, the commands on the left are unspecified; those on 8
the right report on utilities `a` and `c`, respectively. 8

```
time a | b | c            { time a } | b | c          8
a | b | time c            a | b | (time c)            8
```

5.29.3 Options

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

The following option shall be supported by the implementation:

`−p` Write the timing output to standard error in the format shown in 7
5.29.6.2. 7