ChorusOS man pages section 3TELD: Telnet Services

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part No: 806-3336
December 10, 1999

Contents

PREFACE

Overview

A man page is provided for both the naive user, and sophisticated user who is familiar with the ChorusOS™ operating system and is in need of on-line information. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

The following is a list of sections in the ChorusOS man pages and the information it references:

- *Section 1CC: User Utilities; Host and Target Utilities*
- *Section 1M: System Management Utilities*
- *Section 2DL: System Calls; Data Link Services*
- *Section 2K: System Calls; Kernel Services*
- *Section 2MON: System Calls; Monitoring Services*
- *Section 2POSIX: System Calls; POSIX System Calls*
- *Section 2RESTART: System Calls; Hot Restart and Persistent Memory*
- *Section 2SEG: System Calls; Virtual Memory Segment Services*
- *Section 3FTPD: Libraries; FTP Daemon*
- *Section 3M: Libraries; Mathematical Libraries*
- *Section 3POSIX: Libraries; POSIX Library Functions*
- *Section 3RPC: Libraries; RPC Services*
- *Section 3STDC: Libraries; Standard C Library Functions*
- *Section 3TELD: Libraries; Telnet Services*
- *Section 4CC: Files*

- *Section 5FEA: ChorusOS Features and APIs*
- *Section 7P: Protocols*
- *Section 7S: Services*
- *Section 9DDI: Device Driver Interfaces*
- *Section 9DKI: Driver to Kernel Interface*
- *Section 9DRV: Driver Implementations*

ChorusOS man pages are grouped in Reference Manuals, with one reference manual per section.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the `intro` pages for more information and detail about each section, and `man`(1) for more information about man pages in general.

NAME — This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS — This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ] The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.

. . . Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, ‘ "`filename` . . ." .

| Separator. Only one of the arguments separated by this character can be specified at time.

{ } Braces. The options and / or arguments enclosed within braces are

	interdependent, such that everything enclosed must be treated as a unit.
FEATURES	This section provides the list of features which offer an interface. An API may be associated with one or more system features. The interface will be available if one of the associated features has been configured.
DESCRIPTION	This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES.. Interactive commands, subcommands, requests, macros, functions and such, are described under USAGE.
OPTIONS	This lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.
OPERANDS	This section lists the command operands and describes how they affect the actions of the command.
OUTPUT	This section describes the output - standard output, standard error, or output files - generated by the command.
RETURN VALUES	If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.
ERRORS	On failure, most functions place an error code in the global variable `errno` indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.

USAGE	This section is provided as a guidance on use. This section lists special rules, features and commands that require in-depth explanations. The subsections listed below are used to explain built-in functionality: Commands Modifiers Variables Expressions Input Grammar
EXAMPLES	This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%` or if the user must be superuser, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS and USAGE sections.
ENVIRONMENT VARIABLES	This section lists any environment variables that the command or function affects, followed by a brief description of the effect.
EXIT STATUS	This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion and values other than zero for various error conditions.
FILES	This section lists all filenames referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.
SEE ALSO	This section lists references to other man pages, in-house documentation and outside publications.
DIAGNOSTICS	This section lists diagnostic messages with a brief explanation of the condition causing the error.
WARNINGS	This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.
NOTES	This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.

BUGS

This section describes known bugs and wherever possible, suggests workarounds.

CHAPTER

TELNETD Library

10

NAME

Intro – introduction to the TELNETD library

SYNOPSIS

```
#include <arpa/telnetd.h>
```

DESCRIPTION

The TELNETD library provides a set of functions to control and access the server side of a TELNET session. The library functions manage the TELNET protocol, filtering all TELNET commands from the stream of characters sent by the TELNET client.

For each active TELNET session, the library maintains the state associated to the virtual terminal. This state can be modified either by the TELNET client using the TELNET protocol or by the TELNET server using the *telnetdSetTermState* function of the TELNETD library. The *telnetdSetTermState* function can be used to switch the input mode of the TELNET client from line mode to one character at a time mode.

API

The TELNETD library API is summarized in the following table.

Function	Comment
inetAccept	Accept new connections
inetBind	Create and bind sockets to IP ports
inetClient	Get the client IP address
inetClose	Close sockets created by *inetBind*
telnetdFlush	Write out buffered data of a TELNET session
telnetdFree	Free resources associated to a TELNET session
telnetdGetTermState	Get the terminal state of a TELNET session
telnetdInit	Initialize a TELNET session on a new connection
telnetdPasswd	Get the password of the user of a TELNET session
telnetdRead	Read characters from a TELNET session
telnetdReadLine	Read a line of characters from a TELNET session
telnetdSetTermState	Set the terminal state of a TELNET session
telnetdUser	Get the name of the user of a TELNET session
telnetdWrite	Write characters on a TELNET session

As these functions rely on the POSIX socket API, their use should be restricted to a ChorusOS system configured with the enabled POSIX_SOCKETS feature.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

Name	Description
`inetAccept`(3TELD)	wait for a new INET connection
`inetBind`(3TELD)	bind, close INET sockets
`inetClient`(3TELD)	See `inetAccept`(3TELD)
`inetClose`(3TELD)	See `inetBind`(3TELD)
`telnetdFlush`(3TELD)	See `telnetdWrite`(3TELD)
`telnetdFree`(3TELD)	See `telnetdInit`(3TELD)
`telnetdGetTermState`(3TELD)	get or set TELNET terminal state
`telnetdInit`(3TELD)	initialize or free a TELNET session
`telnetdPasswd`(3TELD)	See `telnetdUser`(3TELD)
`telnetdRead`(3TELD)	read from a TELNET session
`telnetdReadLine`(3TELD)	read a line of characters from a TELNET session
`telnetdSetTermState`(3TELD)	See `telnetdGetTermState`(3TELD)
`telnetdUser`(3TELD)	TELNET session authentication
`telnetdWrite`(3TELD)	write or flush a TELNET session

NAME

inetAccept, inetClient – wait for a new INET connection

SYNOPSIS

#include <arpa/telnetd.h>
int **inetAccept**(inetSocket * *sockets*, int *count*, int * *connectionSocket*, int * *connectionPort*);

int **inetClient**(int *connectionSocket*, struct in_addr * *clientInAddr*);

DESCRIPTION

The *inetAccept* function waits for a new `INET` connection.

The *sockets* argument is an array of *count inetSocket* data structures describing sockets previously created and bound to IP ports by *inetBind* .

The *connectionSocket* argument is a result parameter filled with the file descriptor of the new socket created for the new `INET` connection.

The *connectionPort* argument is also a result parameter that is filled with the IP port number on which the client initiated the connection.

The *telnetdInit* function may be invoked later on *connectionSocket* to initialize a `TELNET` session. This session can be controlled by other functions of the `TELNETD` library such as *telnetdUser* , *telnetdRead* or *telnetdSetTermState* .

However, the application may also manage the new connection directly using the low level POSIX socket API. In particular, it can close the socket if it elects to reject the connection (see *close* (2POSIX)), or read all data received on the connection including `TELNET` commands (see *read* (2POSIX)).

The *inetClient* function fills the *clientInAddr* argument with the Internet address of the `TELNET` client connected to the *connectionSocket* socket.

RETURN VALUE

Upon successful completion, *inetAccept* returns a value of zero. Otherwise it returns -1 and sets *errno* to indicate the error.

ERRORS

errno is set to EINVAL if one of the *socket* members of the *inetSocket* data structures is still initialized to *TD_INVALID_SOCKET* .

inetAccept may also fail and set *errno* to any of the errors specified for the *accept* (2POSIX) and *select* (2POSIX) functions.

inetClient may fail and set *errno* to any of the errors specified for the *getpeername* (2POSIX) function.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`inetBind`(3TELD) , `inetClient`(3TELD) , `telnetdInit`(3TELD)

NAME

inetBind, inetClose – bind, close INET sockets

SYNOPSIS

```
#include <arpa/telnetd.h>
int inetBind(inetSocket * sockets, int count);

int inetClose(inetSocket * sockets, int count);
```

DESCRIPTION

The *inetBind* function creates one stream socket and binds it to every IP port from which the server will listen.

The *sockets* parameter is an array of *count inetSocket* data structures. Each *inetSocket* structure has the following members:

```
int     port ;     /* IP port */
int     socket ;   /* socket bound to the port  */
```

where *port* must be a valid IP port number and *socket* must be initialized to *TD_INVALID_SOCKET* . Upon successful return from the call, *socket* will hold the file descriptor of the new socket bound to the IP port.

The *inetClose* function closes all sockets which were created by *inetBind* . It does not close sockets returned by *inetAccept* and does not release any memory allocated by *telnetdInit* for a new TELNET session; this must be done explicitly by closing the socket and calling *telnetdFree* on active TELNET sessions.

RETURN VALUE

Upon successful completion, these functions return a value of zero. Otherwise they return -1 and set *errno* to indicate the error.

ERRORS

errno is set to EINVAL if one of the *socket* members of the *inetSocket* data structures passed to *inetBind* is not initialized to *TD_INVALID_SOCKET* .

The *inetBind* function may also fail and set *errno* to any of the errors specified for the *socket* (2POSIX), and *bind* (2POSIX) functions.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`inetAccept`(3TELD)

NAME

inetAccept, inetClient – wait for a new INET connection

SYNOPSIS

#include <arpa/telnetd.h>
int **inetAccept**(inetSocket * *sockets*, int *count*, int * *connectionSocket*, int * *connectionPort*);

int **inetClient**(int *connectionSocket*, struct in_addr * *clientInAddr*);

DESCRIPTION

The *inetAccept* function waits for a new INET connection.

The *sockets* argument is an array of *count inetSocket* data structures describing sockets previously created and bound to IP ports by *inetBind* .

The *connectionSocket* argument is a result parameter filled with the file descriptor of the new socket created for the new INET connection.

The *connectionPort* argument is also a result parameter that is filled with the IP port number on which the client initiated the connection.

The *telnetdInit* function may be invoked later on *connectionSocket* to initialize a TELNET session. This session can be controlled by other functions of the TELNETD library such as *telnetdUser* , *telnetdRead* or *telnetdSetTermState* .

However, the application may also manage the new connection directly using the low level POSIX socket API. In particular, it can close the socket if it elects to reject the connection (see *close* (2POSIX)), or read all data received on the connection including TELNET commands (see *read* (2POSIX)).

The *inetClient* function fills the *clientInAddr* argument with the Internet address of the TELNET client connected to the *connectionSocket* socket.

RETURN VALUE

Upon successful completion, *inetAccept* returns a value of zero. Otherwise it returns -1 and sets *errno* to indicate the error.

ERRORS

errno is set to EINVAL if one of the *socket* members of the *inetSocket* data structures is still initialized to *TD_INVALID_SOCKET* .

inetAccept may also fail and set *errno* to any of the errors specified for the *accept* (2POSIX) and *select* (2POSIX) functions.

inetClient may fail and set *errno* to any of the errors specified for the *getpeername* (2POSIX) function.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

inetBind(3TELD) , inetClient(3TELD) , telnetdInit(3TELD)

NAME

inetBind, inetClose – bind, close INET sockets

SYNOPSIS

```
#include <arpa/telnetd.h>
int inetBind(inetSocket * sockets, int count);

int inetClose(inetSocket * sockets, int count);
```

DESCRIPTION

The *inetBind* function creates one stream socket and binds it to every IP port from which the server will listen.

The *sockets* parameter is an array of *count inetSocket* data structures. Each *inetSocket* structure has the following members:

```
int     port ;     /* IP port */
int     socket ;   /* socket bound to the port  */
```

where *port* must be a valid IP port number and *socket* must be initialized to *TD_INVALID_SOCKET* . Upon successful return from the call, *socket* will hold the file descriptor of the new socket bound to the IP port.

The *inetClose* function closes all sockets which were created by *inetBind* . It does not close sockets returned by *inetAccept* and does not release any memory allocated by *telnetdInit* for a new TELNET session; this must be done explicitly by closing the socket and calling *telnetdFree* on active TELNET sessions.

RETURN VALUE

Upon successful completion, these functions return a value of zero. Otherwise they return -1 and set *errno* to indicate the error.

ERRORS

errno is set to EINVAL if one of the *socket* members of the *inetSocket* data structures passed to *inetBind* is not initialized to *TD_INVALID_SOCKET* .

The *inetBind* function may also fail and set *errno* to any of the errors specified for the *socket* (2POSIX), and *bind* (2POSIX) functions.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`inetAccept`(3TELD)

NAME

telnetdWrite, telnetdFlush – write or flush a TELNET session

SYNOPSIS

#include <arpa/telnetd.h>
int **telnetdWrite**(telnetdSessionHandle * *hdl*, void * *buf*, unsigned int *nchar*);

int **telnetdFlush**(telnetdSessionHandle * *hdl*);

DESCRIPTION

The *telnetdWrite* function attempts to write *nchar* characters from the buffer pointed to by *buf* to the TELNET session designated by the *hdl* handle.

If the TELNET connection cannot accept data immediatly and the connection socket has O_NONBLOCK set, *telnetdWrite* returns -1 and sets *errno* to EAGAIN. Otherwise, *telnetdWrite* blocks until data can be accepted.

The O_NONBLOCK option may be set for the connection socket by calling *fcntl* (2POSIX) after the TELNET session has been initialized by *telnetdInit*.

The *telnetdFlush* function causes any buffered data waiting to be written for the TELNET session designated by the *hdl* handle to be written to the TELNET connection socket.

RETURN VALUE

Upon successful completion, *telnetdWrite* returns the number of characters actually written; this number may be less than *nchar*. Upon successful completion, *telnetdFlush* returns 0. In case of failure, both functions return -1 and set *errno* to indicate the error.

ERRORS

errno is set to EINVAL if *telnetdFlush* or *telnetdWrite* is called with an invalid *hdl* handle.

The *telnetdWrite* and *telnetdFlush* functions may fail and set *errno* to any of the errors specified for the *write* (2POSIX) function.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

telnetdGetTermState(3TELD), telnetdRead(3TELD)

NAME

telnetdInit, telnetdFree – initialize or free a TELNET session

SYNOPSIS

#include <arpa/telnetd.h>
int **telnetdInit**(int *connectionSocket*, telnetdSessionHandle * *hdl*);

int **telnetdFree**(telnetdSessionHandle * *hdl*);

DESCRIPTION

The *telnetdInit* function initializes a TELNET session for the *connectionSocket* socket obtained by *inetAccept* .

The *hdl* argument is a result parameter filled with the session handle of the new TELNET session. This handle must be used as an argument to other functions of the TELNETD library to designate the new TELNET session.

The *telnetdInit* function allocates memory to maintain the state of the TELNET session. It also sends TELNET options to the client side indicating the ability to do *remote echo* of characters, and to *suppress go ahead* . The state of the virtual terminal is configured to operate in line mode (see *telnetdSetTermState* (3TELD)). The new TELNET session can perform the following functions: *echo* , *binary* , *suppress go ahead* , and *timing mark* . It allows the remote client to perform the following functions: *binary* and *suppress go ahead* .

The *telnetdFree* function frees all memory allocated by *telnetdInit* for the TELNET session designated by its *hdl* handle. The TELNET connection socket associated with the TELNET session is not closed by *telnetdFree* .

RETURN VALUE

Upon successful completion, these functions return a value of zero. Otherwise they return -1 and set *errno* to indicate the error.

ERRORS

The *telnetdInit* function may fail and set *errno* to any of the errors specified for the *malloc* (3STDC), *read* (2POSIX) and *write* (2POSIX) functions.

The *errno* file is set to EINVAL if *telnetdFree* is called with an invalid *hdl* handle.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`telnetdFree`(3TELD) , `telnetdPasswd`(3TELD) , `telnetdRead`(3TELD) , `telnetdUser`(3TELD) , `telnetdWrite`(3TELD)

NAME

telnetdGetTermState, telnetdSetTermState – get or set TELNET terminal state

SYNOPSIS

```
#include <arpa/telnetd.h>
int telnetdGetTermState(telnetdSessionHandle * hdl, telnetdTermState * termState);

int telnetdSetTermState(telnetdSessionHandle * hdl, telnetdTermState * termState);
```

DESCRIPTION

The *telnetdGetTermState* function returns to the *termState* argument the current state of the virtual terminal of the TELNET session designated by the *hdl* handle.

The *telnetdSetTermState* function sets the state of the virtual terminal of the TELNET session designated by the *hdl* handle to the value of the *termState* argument .

The *termState* argument is a pointer to a data structure which has the following member:

```
int     state ; /* terminal state  */
```

where *state* is constructed by or-ing flags from the following list:

TERMSTATE_LINEMODE	client input mode is set to line mode (otherwise client input mode is one character at a time)
TERMSTATE_ECHO	client echoing is enabled (valid only in line mode)
TERMSTATE_EDIT	client line editing is enabled (valid only in line mode)
TERMSTATE_TRAPSIG	client signal trapping is enabled (valid only in line mode)
TERMSTATE_SOFTTAB	client tab expansion is enabled (valid only in line mode)
TERMSTATE_LITECHO	typed control characters are echoed literally by the client (valid only in line mode)
TERMSTATE_CRNL	map CR to NL on input (valid only in line mode)
TERMSTATE_BININ	binary on input.
TERMSTATE_BINOUT	binary on output.
TERMSTATE_FLOW	client flow control is enabled.

`TERMSTATE_RESTARTANY` restart flow on any character.

At initialization time, *telnetdInit* negotiates the following terminal state with the TELNET client:

```
TERMSTATE_FLOW | TERMSTATE_LINEMODE | TERMSTATE_CRNL |
 TERMSTATE_ECHO | TERMSTATE_EDIT | TERMSTATE_TRAPSIG
```

RETURN VALUE

Upon successful completion, the functions return a value of zero. Otherwise they return -1 and set *errno* to indicate the error.

ERRORS

errno is set to EINVAL if one of the functions is called with an invalid *hdl* handle.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`telnetdInit`(3TELD)

NAME

telnetdInit, telnetdFree – initialize or free a TELNET session

SYNOPSIS

```
#include <arpa/telnetd.h>
int telnetdInit(int connectionSocket, telnetdSessionHandle * hdl);

int telnetdFree(telnetdSessionHandle * hdl);
```

DESCRIPTION

The *telnetdInit* function initializes a TELNET session for the *connectionSocket* socket obtained by *inetAccept* .

The *hdl* argument is a result parameter filled with the session handle of the new TELNET session. This handle must be used as an argument to other functions of the TELNETD library to designate the new TELNET session.

The *telnetdInit* function allocates memory to maintain the state of the TELNET session. It also sends TELNET options to the client side indicating the ability to do *remote echo* of characters, and to *suppress go ahead* . The state of the virtual terminal is configured to operate in line mode (see *telnetdSetTermState* (3TELD)). The new TELNET session can perform the following functions: *echo* , *binary* , *suppress go ahead* , and *timing mark* . It allows the remote client to perform the following functions: *binary* and *suppress go ahead* .

The *telnetdFree* function frees all memory allocated by *telnetdInit* for the TELNET session designated by its *hdl* handle. The TELNET connection socket associated with the TELNET session is not closed by *telnetdFree* .

RETURN VALUE

Upon successful completion, these functions return a value of zero. Otherwise they return -1 and set *errno* to indicate the error.

ERRORS

The *telnetdInit* function may fail and set *errno* to any of the errors specified for the *malloc* (3STDC), *read* (2POSIX) and *write* (2POSIX) functions.

The *errno* file is set to EINVAL if *telnetdFree* is called with an invalid *hdl* handle.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`telnetdFree`(3TELD) , `telnetdPasswd`(3TELD) , `telnetdRead`(3TELD) , `telnetdUser`(3TELD) , `telnetdWrite`(3TELD)

NAME

telnetdUser, telnetdPasswd – TELNET session authentication

SYNOPSIS

```
#include <arpa/telnetd.h>
int telnetdUser(telnetdSessionHandle * hdl, char * buf, unsigned int nchar);

int telnetdPasswd(telnetdSessionHandle * hdl, char * buf, unsigned int nchar);
```

DESCRIPTION

The *telnetdUser* function requests the name of the user connected to the TELNET session designated by the *hdl* handle. It then copies this name into the buffer pointed to by *buf* . No more than *nchar* characters are copied to *buf* , and the last character is always the null character.

The *telnetdPasswd* function requests the password of the user connected to the TELNET session designated by the *hdl* handle. It then copies this password into the buffer pointed to by *buf* . No more than *nchar* characters are copied to *buf* , the last character is always the null character. Echoing is turned off while the password is entered by the user.

RETURN VALUE

Upon successful completion, the functions return a value of zero. Otherwise they return -1 and set *errno* to indicate the error.

ERRORS

errno is set to EINVAL if one of the functions is called with an invalid *hdl* handle.

The *telnetdUser* and *telnetdPasswd* functions may fail and set *errno* to any of the errors specified for the *telnetdReadLine* (3TELD) function.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`telnetdReadLine`(3TELD)

NAME

telnetdRead – read from a TELNET session

SYNOPSIS

#include <arpa/telnetd.h>
int **telnetdRead**(telnetdSessionHandle **hdl*, void **buf*, unsigned int *nchar*);

DESCRIPTION

The *telnetdRead* function attempts to read *nchar* characters from the TELNET session designated by the *hdl* handle into the buffer pointed to by *buf*.

While reading characters from the TELNET connection socket, *telnetdRead* filters and executes all TELNET commands sent by the TELNET client. Some of these commands may change the state of the virtual terminal associated with this session (for example, change the client input mode).

The session handle *hdl* is a pointer to a *telnetdSessionHandle* data structure which has the following members:

```
int     socket ;     /* TELNET connection socket */
int     newState ;   /* virtual terminal state flag */
```

where *socket* is the socket associated with the TELNET session. The *newState* flag is set to 1 by *telnetRead* if the state of the virtual terminal changed during the *telnetdRead* operation.

If the TELNET session has no data available and the connection socket has O_NONBLOCK set, *telnetdRead* returns -1 and sets *errno* to EAGAIN. Otherwise, *telnetdRead* blocks until data becomes available or the state of the virtual terminal changes.

The O_NONBLOCK option may be set for the connection socket by calling *fcntl*(2POSIX) after the TELNET session has been initialized using *telnetdInit*.

On return of *telnetdRead* the application should check whether the state of the virtual terminal has changed, and get the new state by calling *telnetdGetTermState*. Calling *telnetdGetTermState* returns the new terminal state; it also clears the *newState* flags of the session handle and session descriptor kept internally by the TELNETD library.

RETURN VALUE

Upon successful completion, *telnetdRead* returns the number of characters actually read and placed in the buffer; this number may be less than *nchar*. Otherwise it returns -1 and sets *errno* to indicate the error.

ERRORS

The *errno* file is set to EINVAL if *telnetdRead* is called with an invalid *hdl* handle.

The *telnetdRead* function may fail and set *errno* to any of the errors specified for the *read*(2POSIX) and *write*(2POSIX) functions.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO `telnetdGetTermState`(3TELD), `telnetdReadLine`(3TELD), `telnetdWrite`(3TELD)

NAME

telnetdReadLine – read a line of characters from a TELNET session

SYNOPSIS

```
#include <arpa/telnetd.h>
int telnetdReadLine(telnetdSessionHandle *hdl, void *buf, unsigned int nchar, int
echo, char *prompt);
```

DESCRIPTION

The *telnetdReadLine* function attempts to read a line of characters from the TELNET session designated by the *hdl* handle into the buffer pointed to by *buf*. A line is delimited by a NEWLINE (ASCII LF) character. This means that *telnetdReadLine* will not complete until an entire line has been typed. However, *nchar* indicates the maximum number of characters to be read. Any number of characters may be requested in *telnetdReadLine*, even one, without losing information. Also, no matter how many characters are requested by the call, a maximum of one line will be returned.

If the value of *echo* is not zero, characters read from the TELNET session will be echoed on the user terminal. Otherwise, no echoing will be performed.

If *prompt* is not the NULL pointer, *telnetdReadLine* first writes the string (terminated by a null character) pointed to by *prompt* on the TELNET connection.

If the TELNET session has no data available and the connection socket has O_NONBLOCK set, *telnetdReadLine* returns -1 and sets *errno* to EAGAIN. Otherwise, *telnetdReadLine* blocks until a line of characters becomes available.

As described in *telnetdRead*, the application should check if on return of *telnetdReadLine* the state of the virtual terminal has changed, and get the new state by calling *telnetdGetTermState*.

RETURN VALUE

Upon successful completion, *telnetdReadLine* returns the number of characters actually read and placed in the buffer; this number may be less than *nchar*. Otherwise it returns -1 and sets *errno* to indicate the error.

ERRORS

errno is set to EINVAL if *telnetdReadLine* is called with an invalid *hdl* handle.

The *telnetdReadLine* function may fail and set *errno* to any of the errors specified for the *telnetdRead*(3TELD) and *telnetdWrite*(3TELD) functions.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

telnetdGetTermState(3TELD), telnetdRead(3TELD), telnetdWrite(3TELD)

NAME

telnetdGetTermState, telnetdSetTermState – get or set TELNET terminal state

SYNOPSIS

```
#include <arpa/telnetd.h>
int telnetdGetTermState(telnetdSessionHandle * hdl, telnetdTermState * termState);

int telnetdSetTermState(telnetdSessionHandle * hdl, telnetdTermState * termState);
```

DESCRIPTION

The *telnetdGetTermState* function returns to the *termState* argument the current state of the virtual terminal of the TELNET session designated by the *hdl* handle.

The *telnetdSetTermState* function sets the state of the virtual terminal of the TELNET session designated by the *hdl* handle to the value of the *termState* argument .

The *termState* argument is a pointer to a data structure which has the following member:

```
int     state ; /* terminal state  */
```

where *state* is constructed by or-ing flags from the following list:

TERMSTATE_LINEMODE	client input mode is set to line mode (otherwise client input mode is one character at a time)
TERMSTATE_ECHO	client echoing is enabled (valid only in line mode)
TERMSTATE_EDIT	client line editing is enabled (valid only in line mode)
TERMSTATE_TRAPSIG	client signal trapping is enabled (valid only in line mode)
TERMSTATE_SOFTTAB	client tab expansion is enabled (valid only in line mode)
TERMSTATE_LITECHO	typed control characters are echoed literally by the client (valid only in line mode)
TERMSTATE_CRNL	map CR to NL on input (valid only in line mode)
TERMSTATE_BININ	binary on input.
TERMSTATE_BINOUT	binary on output.
TERMSTATE_FLOW	client flow control is enabled.

TERMSTATE_RESTARTANY restart flow on any character.

At initialization time, *telnetdInit* negotiates the following terminal state with the TELNET client:

```
TERMSTATE_FLOW | TERMSTATE_LINEMODE | TERMSTATE_CRNL |
 TERMSTATE_ECHO | TERMSTATE_EDIT | TERMSTATE_TRAPSIG
```

RETURN VALUE

Upon successful completion, the functions return a value of zero. Otherwise they return -1 and set *errno* to indicate the error.

ERRORS

errno is set to EINVAL if one of the functions is called with an invalid *hdl* handle.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

telnetdInit(3TELD)

NAME

telnetdUser, telnetdPasswd – TELNET session authentication

SYNOPSIS

#include <arpa/telnetd.h>
int **telnetdUser**(telnetdSessionHandle * *hdl*, char * *buf*, unsigned int *nchar*);

int **telnetdPasswd**(telnetdSessionHandle * *hdl*, char * *buf*, unsigned int *nchar*);

DESCRIPTION

The *telnetdUser* function requests the name of the user connected to the TELNET session designated by the *hdl* handle. It then copies this name into the buffer pointed to by *buf* . No more than *nchar* characters are copied to *buf* , and the last character is always the null character.

The *telnetdPasswd* function requests the password of the user connected to the TELNET session designated by the *hdl* handle. It then copies this password into the buffer pointed to by *buf* . No more than *nchar* characters are copied to *buf* , the last character is always the null character. Echoing is turned off while the password is entered by the user.

RETURN VALUE

Upon successful completion, the functions return a value of zero. Otherwise they return -1 and set *errno* to indicate the error.

ERRORS

errno is set to EINVAL if one of the functions is called with an invalid *hdl* handle.

The *telnetdUser* and *telnetdPasswd* functions may fail and set *errno* to any of the errors specified for the *telnetdReadLine* (3TELD) function.

ATTRIBUTES

See `attributes`(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

`telnetdReadLine`(3TELD)

NAME

telnetdWrite, telnetdFlush – write or flush a TELNET session

SYNOPSIS

#include <arpa/telnetd.h>
int **telnetdWrite**(telnetdSessionHandle * *hdl*, void * *buf*, unsigned int *nchar*);

int **telnetdFlush**(telnetdSessionHandle * *hdl*);

DESCRIPTION

The *telnetdWrite* function attempts to write *nchar* characters from the buffer pointed to by *buf* to the TELNET session designated by the *hdl* handle.

If the TELNET connection cannot accept data immediatly and the connection socket has O_NONBLOCK set, *telnetdWrite* returns -1 and sets *errno* to EAGAIN . Otherwise, *telnetdWrite* blocks until data can be accepted.

The O_NONBLOCK option may be set for the connection socket by calling *fcntl* (2POSIX) after the TELNET session has been initialized by *telnetdInit* .

The *telnetdFlush* function causes any buffered data waiting to be written for the TELNET session designated by the *hdl* handle to be written to the TELNET connection socket.

RETURN VALUE

Upon successful completion, *telnetdWrite* returns the number of characters actually written; this number may be less than *nchar* . Upon successful completion, *telnetdFlush* returns 0. In case of failure, both functions return -1 and set *errno* to indicate the error.

ERRORS

errno is set to EINVAL if *telnetdFlush* or *telnetdWrite* is called with an invalid *hdl* handle.

The *telnetdWrite* and *telnetdFlush* functions may fail and set *errno* to any of the errors specified for the *write* (2POSIX) function.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

SEE ALSO

telnetdGetTermState(3TELD) , telnetdRead(3TELD)

Index