N E T X™

Telnet Protocol (Telnet)

User Guide

Express Logic, Inc.

858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259

www.expresslogic.com

Part Number: 000-1055
Revision 5.10

Contents

Chapter 1

Introduction to Telnet

The Telnet Protocol (Telnet) is a protocol designed for transferring commands and responses between two nodes on the Internet. Telnet is a simple protocol that utilizes reliable Transmission Control Protocol (TCP) services to perform its transfer function. Because of this, Telnet is a highly reliable transfer protocol. Telnet is also one of the most used application protocols.

Telnet Requirements

In order to function properly, the NetX Telnet package requires that a NetX IP instance has already been created. In addition, TCP must be enabled on that same IP instance. The Telnet Client portion of the NetX Telnet package has no further requirements.

The Telnet Server portion of the NetX Telnet package has one additional requirement. It requires complete access to TCP *well-known port 23* for handling all Client Telnet requests.

Telnet Constraints

The NetX Telnet protocol implements the Telnet standard. However, the interpretation and response of Telnet commands, indicated by a byte with the value of 255, is the responsibility of the application. The various Telnet commands and command parameters are defined in the *nx_telnet_client.h* and *nx_telnet_server.h* files.

Telnet Communication

As mentioned previously, the Telnet Server utilizes the *well-known TCP port 23* to field Client requests. Telnet Clients may use any available TCP port.

Telnet Authentication

Telnet authentication is the responsibility of the application's Telnet Server callback function. The application's Telnet Server "new connection" callback would typically prompt the Client for name and/or password. The Client would then be responsible for providing the information. The Server would then process the information in the "receive data" callback. This is where the application Server code would have to authenticate the information and decide whether or not it is valid.

Telnet New Connection Callback

The NetX Telnet Server calls the application specified callback function whenever a new Telnet Client request is received. The application specifies the callback function when the Telnet Server is created via the **nx_telnet_server_create** function. Typical actions of the "new connection" callback include sending a banner or prompt to the Client. This could very well include a prompt for login information.

The format of the application "new connection" callback routine is very simple and is defined below:

```
void  telnet_new_connection(NX_Telnet_SERVER *server_ptr,
                            UINT logical_connection);
```

The input parameters are defined as follows:

Parameter	Meaning
server_ptr	Pointer to the calling Telnet Server.
logical_connection	The internal logical connection for the Telnet Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_Telnet_MAX_CLIENTS-1.

Telnet Receive Data Callback

The NetX Telnet Server calls the application specified callback function whenever a new Telnet Client data is received. The application specifies

the callback function when the Telnet Server is created via the **nx_telnet_server_create** function. Typical actions of the "new connection" callback include echoing the data back and/or parsing the data and providing data as a result of interpreting a command from the client.

Note that this callback routine must also release the supplied packet.

The format of the application "receive data" callback routine is very simple and is defined below:

```
void  telnet_receive_data(NX_Telnet_SERVER *server_ptr,
             UINT logical_connection, NX_PACKET *packet_ptr);
```

The input parameters are defined as follows:

Parameter	Meaning
server_ptr	Pointer to the calling Telnet Server.
logical_connection	The internal logical connection for the Telnet Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_Telnet_MAX_CLIENTS-1.
packet_ptr	Pointer to packet containing the data from the Client.

Telnet End Connection Callback

The NetX Telnet Server calls the application specified callback function whenever a Telnet Client ends the connection. The application specifies the callback function when the Telnet Server is created via the **nx_telnet_server_create** function. Typical actions of the "end connection" callback include cleaning up any Client specific data structures associated with the logical connection.

The format of the application "end connection" callback routine is very simple and is defined below:

```
void  telnet_end_connection(NX_Telnet_SERVER *server_ptr,
                   UINT logical_connection);
```

The input parameters are defined as follows:

Parameter	Meaning
server_ptr	Pointer to the calling Telnet Server.
logical_connection	The internal logical connection for the Telnet Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_Telnet_MAX_CLIENTS-1.

Telnet Option Negotiation

The NetX Telnet Server supports a limited set of Telnet options, `Echo` and `Suppress Go Ahead`.

To enable this feature the NX_TELNET_SERVER_OPTION_DISABLE must not be defined. By default it is not defined. The Telnet Server creates a packet pool in the *nx_telnet_server_create* service from which it allocates packets for sending telnet options requests to the Client. See "Configuration Options" for setting the packet payload (NX_TELNET_SERVER_PACKET_PAYLOAD) and packet pool size (NX_TELNET_SERVER_PACKET_POOL_SIZE**)** for this packet pool. It will delete this packet pool when the *nx_telnet_server_delete* service is called.

Upon making a connection with the Telnet Client, it will send out this set of telnet options to the Client if it has not received option requests from the Client:

```
will echo

dont echo

will sga
```

When it receives Telnet data from the Client, the Telnet Server checks if the first byte is the "IAC" code. If so, it will process all the options in the Client packet. Options not in the list above are ignored.

By default, the Telnet Server creates its own internal packet pool if NX_TELNET_SERVER_OPTION_DISABLE is not defined and it needs to transmit Telnet option commands. The Telnet Server packet pool is defined by

NX_TELNET_SERVER_PACKET_PAYLOAD and NX_TELNET_SERVER_PACKET_POOLSIZE. If, however, NX_TELNET_SERVER_USER_CREATE_PACKET_POOL is defined, the application must create the Telnet Server packet pool and set it as the Telnet Server packet pool by calling *_nx_telnet_server_packet_pool_set*. See Chapter 3 "Description of Telnet Services" for more details about this function.

Telnet Multi-Thread Support

The NetX Telnet Client services can be called from multiple threads simultaneously. However, read or write requests for a particular Telnet Client instance should be done in sequence from the same thread.

Telnet RFCs

NetX Telnet is compliant with RFC854 and related RFCs.

Chapter 2

Installation and Use of Telnet

This chapter contains a description of various issues related to installation, setup, and usage of the NetX Telnet component.

Product Distribution

Telnet for NetX is shipped on a single CD-ROM compatible disk. The package includes three source files, two include files, and a PDF file that contains this document, as follows:

nx_telnet_client.h	Header file for Telnet Client for NetX
nx_telnet_client.c	C Source file for Telnet Client for NetX
nx_telnet_server.h	Header file for Telnet Server for NetX
nx_telnet_server.c	C Source file for Telnet Server for NetX
nx_telnet.pdf	PDF description of Telnet for NetX
demo_netx_telnet.c	NetX Telnet demonstration

Telnet Installation

In order to use Telnet for NetX, the entire distribution mentioned previously should be copied to the same directory where NetX is installed. For example, if NetX is installed in the directory "*\threadx\arm7\green*" then the *nx_telnet_client.h*, *nx_telnet_client.c*, *nx_telnet_server.c* and *nx_telnet_server.h* files should be copied into this directory.

Using Telnet

Using Telnet for NetX is easy. Basically, the application code must include the header files after it includes *tx_api.h* and *nx_api.h*, in order to use ThreadX and NetX. Once the header files are included, the application code is then able to make the Telnet function calls specified later in this guide. The application must also include *nx_telnet_client.c* and *nx_telnet_server.c* in the build process. These files must be compiled in the same manner as other application files and its object form must be linked along with the files of the application. This is all that is required to use NetX Telnet.

If no Telnet Client capabilities are required, the *nx_telnet_client.c* file may be omitted.

Note also that because Telnet utilizes NetX TCP services, TCP must be enabled with the *nx_tcp_enable* call prior to using Telnet.

Small Example System

An example of how easy it is to use NetX Telnet is described in Figure 1.1 that appears below. In this example, the Telnet include file *nx_telnet_client.h and nx_telnet_server.h* are brought in at line 7. Next, the Telnet Server is created in "*tx_application_define*" at line 112. Note that the Telnet Server control block "*Server*" was defined as a global variable at line 24 previously. After successful creation, an Telnet Server is started at line 121. At line 138 the Telnet Client is created. And finally, the Client sends a character at line 160 and reads the character back at line 182.

```
001 /* This is a small demo of Telnet on the high-performance NetX TCP/IP stack.
002    This demo relies on ThreadX and NetX to show a simple Telnet connection,
003    send, server echo, and then disconnection from the Telnet server.  */
004
005 #include  "tx_api.h"
006 #include  "nx_api.h"
007 #include  "nx_telnet_client.h"
008 #include  "nx_telnet_server.h"
009 #define    DEMO_STACK_SIZE         4096
010
011
012 /* Define the ThreadX and NetX object control blocks...  */
013
014 TX_THREAD                thread_0;
015 TX_THREAD                thread_1;
016 NX_PACKET_POOL           pool_0;
017 NX_PACKET_POOL           pool_1;
018 NX_IP                    ip_0;
019 NX_IP                    ip_1;
020
021
022 /* Define Telnet objects.  */
023
024 NX_Telnet_SERVER         my_server;
025 NX_Telnet_CLIENT         my_client;
026
027 /* Define the counters used in the demo application...  */
028
029 ULONG                    error_counter;
030
031
032 /* Define function prototypes.  */
033
034 void    thread_0_entry(ULONG thread_input);
035 void    _nx_ram_network_driver(struct NX_IP_DRIVER_STRUCT *driver_req);
036
037
038 /* Define the application's Telnet Server callback routines.  */
039
040 void    telnet_new_connection(NX_Telnet_SERVER *server_ptr,
041                                       UINT logical_connection);
042 void    telnet_receive_data(NX_Telnet_SERVER *server_ptr,
043                    UINT logical_connection, NX_PACKET *packet_ptr);
044 void    telnet_connection_end(NX_Telnet_SERVER *server_ptr,
045                    UINT logical_connection);
046
047
048 /* Define main entry point.  */
049
050 int main()
051 {
```

```
052
053        /* Enter the ThreadX kernel.  */
054        tx_kernel_enter();
055 }
056
057
058 /* Define what the initial system looks like.  */
059 void    tx_application_define(void *first_unused_memory)
060 {
061
062 UINT    status;
063 CHAR    *pointer;
064
065
066        /* Setup the working pointer.  */
067        pointer =  (CHAR *) first_unused_memory;
068
069        /* Create the main thread.  */
070        tx_thread_create(&thread_0, "thread 0", thread_0_entry, 0,
071                pointer, DEMO_STACK_SIZE,
072                2, 2, TX_NO_TIME_SLICE, TX_AUTO_START);
073        pointer =  pointer + DEMO_STACK_SIZE;
074
075        /* Initialize the NetX system.  */
076        nx_system_initialize();
077
078        /* Create packet pool.  */
079        nx_packet_pool_create(&pool_0, "NetX Packet Pool 0",
080                                              600, pointer, 8192);
081        pointer = pointer + 8192;
082
083        /* Create an IP instance.  */
084        nx_ip_create(&ip_0, "NetX IP Instance 0", IP_ADDRESS(1, 2, 3, 4),
085                        0xFFFFFF00UL, &pool_0, _nx_ram_network_driver,
086                        pointer, 4096, 1);
087        pointer =  pointer + 4096;
088
089        /* Create another packet pool. */
090        nx_packet_pool_create(&pool_1, "NetX Packet Pool 1",600,pointer,8192);
091        pointer = pointer + 8192;
092
093        /* Create another IP instance.  */
094        nx_ip_create(&ip_1, "NetX IP Instance 1", IP_ADDRESS(1, 2, 3, 5),
095                        0xFFFFFF00UL, &pool_1, _nx_ram_network_driver,
096                        pointer, 4096, 1);
097        pointer = pointer + 4096;
098
099        /* Enable ARP and supply ARP cache memory for IP Instance 0.  */
100        nx_arp_enable(&ip_0, (void *) pointer, 1024);
101        pointer = pointer + 1024;
102
103        /* Enable ARP and supply ARP cache memory for IP Instance 1.  */
104        nx_arp_enable(&ip_1, (void *) pointer, 1024);
105        pointer = pointer + 1024;
106
107        /* Enable TCP processing for both IP instances.  */
108        nx_tcp_enable(&ip_0);
109        nx_tcp_enable(&ip_1);
110
111        /* Create the NetX Telnet Server.  */
112        status = nx_telnet_server_create(&my_server, "Telnet Server", &ip_0,
113                        pointer, 2048, telnet_new_connection,
114                        telnet_receive_data, telnet_connection_end);
115
116        /* Check for errors.  */
117        if (status)
118            error_counter++;
119
120        /* Start the Telnet Server.  */
121        status = nx_telnet_server_start(&my_server);
122
123        /* Check for errors.  */
124        if (status)
125            error_counter++;
126 }
127
128
129 /* Define the test thread.  */
130 void    thread_0_entry(ULONG thread_input)
131 {
132
```

```
133 NX_PACKET    *my_packet;
134 UINT         status;
135
136
137     /* Create a TELENT client instance.  */
138     status =  nx_telnet_client_create(&my_client, "My Telnet Client",
139                                                   &ip_1, 600);
140
141     /* Check status.  */
142     if (status)
143         error_counter++;
144
145     /* Connect the Telnet client to the Telnet Server at port 23.  */
146     status =  nx_telnet_client_connect(&my_client, IP_ADDRESS(1,2,3,4), 23, 50);
147
148     /* Check status.  */
149     if (status)
150         error_counter++;
151
152     /* Allocate a packet.  */
153     status =  nx_packet_allocate(&pool_0, &my_packet, NX_TCP_PACKET,
154                                                   NX_WAIT_FOREVER);
155
156     /* Build a simple 1-byte message.  */
157     nx_packet_data_append(my_packet, "a", 1, &pool_0, NX_WAIT_FOREVER);
158
159     /* Send the packet to the Telnet Server.  */
160     status =  nx_telnet_client_packet_send(&my_client, my_packet, 50);
161
162     /* Check status.  */
163     if (status)
164         error_counter++;
165
166     /* Pickup the Server header.  */
167     status =  nx_telnet_client_packet_receive(&my_client, &my_packet, 50);
168
169     /* Check status.  */
170     if (status)
171         error_counter++;
172     else
173     {
174
175         /* At this point the packet should contain the Server's banner
176            message sent by the Server callback function below.  Just
177            release it for this demo.  */
178         nx_packet_release(my_packet);
179     }
180
181     /* Pickup the Server echo of the character.  */
182     status =  nx_telnet_client_packet_receive(&my_client, &my_packet, 50);
183
184     /* Check status.  */
185     if (status)
186         error_counter++;
187     else
188     {
189
190         /* At this point the packet should contain the character 'a' that
191            we sent earlier.  Just release the packet for now.  */
192         nx_packet_release(my_packet);
193     }
194
195     /* Now disconnect form the Telnet Server.  */
196     status =  nx_telnet_client_disconnect(&my_client, 50);
197
198     /* Check status.  */
199     if (status)
200         error_counter++;
201
202     /* Delete the Telnet Client.  */
203     status =  nx_telnet_client_delete(&my_client);
204
205     /* Check status.  */
206     if (status)
207         error_counter++;
208 }
209
210
211 /* This routine is called by the NetX Telnet Server whenever a new Telnet client
212    connection is established.  */
213 void  telnet_new_connection(NX_Telnet_SERVER *server_ptr, UINT logical_connection)
```

```
214 {
215
216 UINT        status;
217 NX_PACKET   *packet_ptr;
218
219     /* Allocate a packet for client greeting. */
220     status =  nx_packet_allocate(&pool_0, &packet_ptr, NX_TCP_PACKET, NX_NO_WAIT);
221
222     /* Determine if we have a packet.  */
223     if (status == NX_SUCCESS)
224     {
225
226         /* Build a banner message and a prompt.  */
227         nx_packet_data_append(packet_ptr,
228                 "**** Welcome to NetX Telnet Server ****\r\n\r\n\r\n\r\n", 45,
229                 &pool_0, NX_NO_WAIT);
230         nx_packet_data_append(packet_ptr, "NETX> ", 6, &pool_0, NX_NO_WAIT);
231
232         /* Send the packet to the client.  */
233         status =  nx_telnet_server_packet_send(server_ptr, logical_connection,
234                                                             packet_ptr, 100);
235
236         if (status)
237             nx_packet_release(packet_ptr);
238     }
239 }
240
241
242 /* This routine is called by the NetX Telnet Server whenever data is
243    present on a Telnet client connection.  */
244 void  telnet_receive_data(NX_Telnet_SERVER *server_ptr, UINT logical_connection,
245                                                     NX_PACKET *packet_ptr)
246 {
247
248 UINT    status;
249 UCHAR   alpha;
250
251
252     /* This demo just echoes the character back and on <cr,lf> sends a new prompt
253        back to the client. A real system would most likely buffer the character(s)
254        received in a buffer associated with the supplied logical connection and
255        process according to it.  */
256
257     /* Just throw away carriage returns.  */
258     if ((packet_ptr -> nx_packet_prepend_ptr[0] == '\r') &&
259                                     (packet_ptr -> nx_packet_length == 1))
260     {
261         nx_packet_release(packet_ptr);
262         return;
263     }
264
265     /* Setup new line on line feed.  */
266     if ((packet_ptr -> nx_packet_prepend_ptr[0] == '\n') ||
267                         (packet_ptr -> nx_packet_prepend_ptr[1] == '\n'))
268     {
269
270         /* Clean up the packet.  */
271         packet_ptr -> nx_packet_length =  0;
272         packet_ptr -> nx_packet_prepend_ptr =
273                         packet_ptr -> nx_packet_data_start + NX_TCP_PACKET;
274         packet_ptr -> nx_packet_append_ptr =
275                         packet_ptr -> nx_packet_data_start + NX_TCP_PACKET;
276
277         /* Build the next prompt.  */
278         nx_packet_data_append(packet_ptr, "\r\nNETX> ", 8, &pool_0, NX_NO_WAIT);
279
280         /* Send the packet to the client.  */
281         status =  nx_telnet_server_packet_send(server_ptr, logical_connection,
282                                                             packet_ptr, 100);
283
284         if (status)
285             nx_packet_release(packet_ptr);
286         return;
287     }
288
289     /* Pickup first character (usually only one from client).  */
290     alpha =  packet_ptr -> nx_packet_prepend_ptr[0];
291
292     /* Echo character.  */
293     status =  nx_telnet_server_packet_send(server_ptr, logical_connection,
294                                                             packet_ptr, 100);
```

```
295
296     if (status)
297          nx_packet_release(packet_ptr);
298
299     /* Check for a disconnection.  */
300     if (alpha == 'q')
301     {
302
303          /* Initiate server disconnection.  */
304          nx_telnet_server_disconnect(server_ptr, logical_connection);
305     }
306 }
307
308
309 /* This routine is called by the NetX Telnet Server whenever the
310    client disconnects.  */
311 void  telnet_connection_end(NX_Telnet_SERVER *server_ptr,
312                                                UINT logical_connection)
313 {
314
315     /* Cleanup any application specific connection or buffer information.  */
316 }
```

Figure 1.1 Example of Telnet use with NetX

Configuration Options

There are several configuration options for building Telnet for NetX. These #defines can be set by the application prior to inclusion of *nx_telnet_server.h*.and *nx_telnet_client.h*

Following is a list of all options, where each is described in detail:

Define	Meaning
NX_DISABLE_ERROR_CHECKING	Defined, this option removes the basic Telnet error checking. It is typically used after the application has been debugged.
NX_TELNET_MAX_CLIENTS	The maximum number of Telnet Clients supported by the Server thread. By default, this value is defined as 4 to specify a maximum of 4 clients at a time.
NX_TELNET_SERVER_PRIORITY	The priority of the Telnet Server thread. By default, this value is defined as 16 to specify priority 16.
NX_TELNET_TOS	Type of service required for the Telnet TCP requests. By default, this value is defined as NX_IP_NORMAL to indicate normal IP packet service.
NX_TELNET_FRAGMENT_OPTION	Fragment enable for Telnet TCP requests. By default, this value is NX_DONT_FRAGMENT to disable Telnet TCP fragmenting.
NX_TELNET_SERVER_WINDOW_SIZE	Server socket window size. By default, this value is 2048 bytes.
NX_TELNET _TIME_TO_LIVE	Specifies the number of routers this packet can pass before it

	is discarded. The default value is set to 0x80.
NX_TELNET _SERVER_TIMEOUT	Specifies the number of ThreadX ticks that internal services will suspend for. The default value is set to 10 seconds.
NX_TELNET_ACTIVITY_TIMEOUT	Specifies the number of seconds that can elapse without any activity before the Server disconnects the Client connection. The default value is set to 600 seconds.
NX_TELNET_TIMEOUT_PERIOD	Specifies the number of seconds between checking for Client activity timeouts. The default value is set to 60 seconds.

NX_TELNET_SERVER_OPTION_DISABLE

Defined, Telnet option negotiation is disabled. By default this option is not defined.

NX_TELNET_SERVER_USER_CREATE_PACKET_POOL

If defined, the Telnet Server packet pool must be created externally. This is only meaningful if NX_TELNET_SERVER_OPTION_DISABLE is not defined. By default this option is not defined and the Telnet Server thread creates its own packet pool.

NX_TELNET_SERVER_PACKET_PAYLOAD

Defines the size of the packet payload created by the Telnet Server for option negotiation. Note that the Telnet Server only creates this packet pool if NX_TELNET_SERVER _OPTION_DISABLE is not defined (Telnet options are

enabled). The default value of
this option is 300.

NX_TELNET_SERVER_PACKET_POOL_SIZE

Defines the size of the Telnet
Server packet pool used for
Telnet negotiations. Note that the
Telnet Server only creates this
packet pool if NX_TELNET_SERVER
_OPTION_DISABLE is not defined
(Telnet options are enabled).
The default value of this option
is 2048 (~5-6 packets).

Chapter 3

Description of Telnet Services

This chapter contains a description of all NetX Telnet services (listed below) in alphabetic order.

In the "Return Values" section in the following API descriptions, values in **BOLD** are not affected by the **NX_DISABLE_ERROR_CHECKING** define that is used to disable API error checking, while non-bold values are completely disabled.

nx_telnet_client_connect
Connect a Telnet Client

nx_telnet_client_create
Create a Telnet Client

nx_telnet_client_delete
Delete a Telnet Client

nx_telnet_client_disconnect
Disconnect a Telnet Client

nx_telnet_client_packet_receive
Receive packet via Telnet Client

nx_telnet_client_packet_send
Send packet via Telnet Client

nx_telnet_server_create
Create a Telnet Server

nx_telnet_server_delete
Delete a Telnet Server

nx_telnet_server_disconnect
Disconnect a Telnet Client

nx_telnet_server_packet_send
Send packet through Client connection

nx_telnet_server_packet_pool_set
Set packet pool as Telnet Server packet pool

nx_telnet_server_start
Start a Telnet Server

nx_telnet_server_stop
Stop a Telnet Server

nx_telnet_client_connect

Connect a Telnet Client

Prototype

```
UINT nx_telnet_client_connect(NX_Telnet_CLIENT *client_ptr,
          ULONG server_ip, UINT server_port, ULONG wait_option);
```

Description

This service attempts to connect the previously created Telnet Client instance to the Server at the specified IP and port.

Input Parameters

client_ptr   Pointer to Telnet Client control block.

server_ip   IP Address of Server.

server_port   TCP Port of Server (Telnet Server is port 23).

wait_option   Defines how long the service will wait for the Telnet Client connect. The wait options are defined as follows:

                timeout value   (0x00000001 through 0xFFFFFFFE)
                TX_WAIT_FOREVER (0xFFFFFFFF)

                Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the Telnet Server responds to the request.

                Selecting a numeric value (1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the Telnet Server response.

Return Values

NX_SUCCESS   (0x00)   Successful Client connect.
NX_TELNET_NOT_DISCONNECTED
                (0xF4) Client already connected.

status		Actual NetX completion status
NX_PTR_ERROR	(0x16)	Invalid Client pointer.
NX_IP_ADDRESS_ERROR		
	(0x21)	Invalid IP address.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Connect the Telnet Client instance "my_client" to the Server at
   IP address 1.2.3.4 and port 23.  */
status =  nx_telnet_client_connect(&my_client, IP_ADDRESS(1,2,3,4), 23, 100);


/* If status is NX_SUCCESS the Telnet Client instance was successfully
   connected to the Telnet Server.  */
```

nx_telnet_client_create

Create a Telnet Client

Prototype

```
UINT nx_telnet_client_create(NX_Telnet_CLIENT *client_ptr,
                CHAR *client_name, NX_IP *ip_ptr, ULONG window_size);
```

Description

This service creates a Telnet Client instance.

Input Parameters

client_ptr   Pointer to Telnet Client control block.

client_name   Name of Client instance.

ip_ptr   Pointer to IP instance.

window_size   Size of TCP receive window for this Client.

Return Values

NX_SUCCESS	(0x00)	Successful Client create.
status		Actual NetX completion status
NX_PTR_ERROR	(0x16)	Invalid Client or IP pointer.

Allowed From

Initialization, Threads

Example

```
/* Create the Telnet Client instance "my_client" on the IP instance "ip_0".  */
status =  nx_telnet_client_create(&my_client, "My Telnet Client", &ip_0, 2048);

/* If status is NX_SUCCESS the Telnet Client instance was successfully
   created.  */
```

nx_telnet_client_delete

Delete a Telnet Client

Prototype

UINT **nx_telnet_client_delete**(NX_Telnet_CLIENT *client_ptr);

Description

This service deletes a previously created Telnet Client instance.

Input Parameters

client_ptr   Pointer to Telnet Client control block.

Return Values

NX_SUCCESS   (0x00)   Successful Client delete.
NX_TELNET_NOT_DISCONNECTED
                                (0xF4)   Client still connected.
NX_PTR_ERROR   (0x16)   Invalid Client pointer.
NX_CALLER_ERROR   (0x11)   Invalid caller of this
                                                service.

Allowed From

Threads

Example

```
/* Delete the Telnet Client instance "my_client".  */
status = nx_telnet_client_delete(&my_client);

/* If status is NX_SUCCESS the Telnet Client instance was successfully
   deleted.  */
```

nx_telnet_client_disconnect

Disconnect a Telnet Client

Prototype

```
UINT nx_telnet_client_disconnect(NX_Telnet_CLIENT *client_ptr,
                                 ULONG wait_option);
```

Description

This service disconnects a previously connected Telnet Client instance.

Input Parameters

client_ptr   Pointer to Telnet Client control block.

wait_option   Defines how long the service will wait for the
Telnet Client disconnect. The wait options are
defined as follows:

> **timeout value**   (0x00000001 through
>   0xFFFFFFFE)
> **TX_WAIT_FOREVER** (0xFFFFFFFF)
>
> Selecting TX_WAIT_FOREVER causes the
> calling thread to suspend indefinitely until the
> Telnet Server responds to the request.
>
> Selecting a numeric value (1-0xFFFFFFFE)
> specifies the maximum number of timer-ticks
> to stay suspended while waiting for the Telnet
> Server response.

Return Values

NX_SUCCESS   (0x00)   Successful Client disconnect.
NX_TELNET_NOT_CONNECTED
                        (0xF3)   Client not connected.
NX_PTR_ERROR   (0x16)   Invalid Client pointer.
NX_CALLER_ERROR   (0x11)   Invalid caller of this service.

Allowed From

Threads

Example

```
/* Disconnect the Telnet Client instance "my_client".  */
status = nx_telnet_client_disconnect(&my_client, 100);


/* If status is NX_SUCCESS the Telnet Client instance was successfully
   disconnected.  */
```

nx_telnet_client_packet_receive

Receive packet via Telnet Client

Prototype

```
UINT nx_telnet_client_packet_receive(NX_Telnet_CLIENT *client_ptr,
                NX_PACKET **packet_ptr, ULONG wait_option);
```

Description

This service receives a packet from the previously connected Telnet Client instance.

Input Parameters

client_ptr   Pointer to Telnet Client control block.

packet_ptr   Pointer to the destination for the received packet.

wait_option   Defines how long the service will wait for the Telnet Client packet receive. The wait options are defined as follows:

>   **timeout value**   (0x00000001 through 0xFFFFFFFE)
>   **TX_WAIT_FOREVER** (0xFFFFFFFF)
>
>   Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the Telnet Server responds to the request.
>
>   Selecting a numeric value (1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the Telnet Server response.

Return Values

NX_SUCCESS   (0x00)   Successful Client packet receive.
status   Actual NetX completion status
NX_PTR_ERROR   (0x16)   Invalid Client or packet pointer.

NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Receive a packet from the Telnet Client instance "my_client".  */
status =  nx_telnet_client_packet_receive(&my_client, &my_packet, 100);

/* If status is NX_SUCCESS the "my_packet" pointer contains data received from
   the Telnet Client connection.  */
```

nx_telnet_client_packet_send

Send packet via Telnet Client

Prototype

```
UINT nx_telnet_client_packet_send(NX_Telnet_CLIENT *client_ptr,
                        NX_PACKET *packet_ptr, ULONG wait_option);
```

Description

This service sends a packet through the previously connected Telnet Client instance.

Input Parameters

client_ptr   Pointer to Telnet Client control block.

packet_ptr   Pointer to the packet to send.

wait_option   Defines how long the service will wait for the Telnet Client packet send. The wait options are defined as follows:

timeout value   (0x00000001 through 0xFFFFFFFE)
TX_WAIT_FOREVER (0xFFFFFFFF)

Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the Telnet Server responds to the request.

Selecting a numeric value (1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the Telnet Server response.

Return Values

NX_SUCCESS   (0x00)   Successful packet send.
NX_PTR_ERROR   (0x16)   Invalid pointer input
NX_CALLER_ERROR   (0x11)   Invalid caller of this
                                   service.

Allowed From

Threads

Example

```
/* Send a packet via the Telnet Client instance "my_client".  */
status =  nx_telnet_client_packet_send(&my_client, my_packet, 100);

/* If status is NX_SUCCESS the packet was successfully sent.  */
```

nx_telnet_server_create

Create a Telnet Server

Prototype

```
UINT nx_telnet_server_create(NX_Telnet_SERVER *server_ptr,
                CHAR *server_name, NX_IP *ip_ptr,
                VOID *stack_ptr, ULONG stack_size,
                void (*new_connection)(struct NX_Telnet_SERVER_STRUCT
                    *telnet_server_ptr, UINT logical_connection),
                void (*receive_data)(struct NX_Telnet_SERVER_STRUCT
                    *telnet_server_ptr, UINT logical_connection,
                    NX_PACKET *packet_ptr),
                void (*connection_end)(struct NX_Telnet_SERVER_STRUCT
                    *telnet_server_ptr, UINT logical_connection));
```

Description

This service creates a Telnet Server instance on the specified IP instance.

Input Parameters

> **server_ptr**   Pointer to Telnet Server control block.
>
> **server_name**   Name of Telnet Server instance.
>
> **ip_ptr**   Pointer to associated IP instance.
>
> **stack_ptr**   Pointer to stack for the internal Server thread.
>
> **sack_size**   Size of the stack, in bytes.
>
> **new_connection**   Application callback routine function pointer. This
>   routine is called whenever a new Telnet Client
>   connection request is detected by the Server.
>
> **receive_data**   Application callback routine function pointer. This
>   routine is called whenever a new Telnet Client
>   data is present on the connection. This routine is
>   responsible for releasing the packet.
>
> **end_connection**   Application callback routine function pointer. This
>   routine is called whenever a Telnet Client
>   connection is disconnected by the Client. The Server
>   can also disconnect via the
>   *nx_telnet_server_disconnect* service described below.

Return Values

NX_SUCCESS	(0x00)	Successful Server delete.
NX_PTR_ERROR	(0x07)	Invalid Server pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Initialization, Threads

Example

```
/* Create a Telnet Server instance "my_server".  */
status =  nx_telnet_server_create(&my_server, "Telnet Server", &ip_0,
                 pointer, 2048, telnet_new_connection, telnet_receive_data,
                 telnet_connection_end);

/* If status is NX_SUCCESS the Telnet Server was successfully created.  */
```

nx_telnet_server_delete

Delete a Telnet Server

Prototype

UINT **nx_telnet_server_delete**(NX_Telnet_SERVER *server_ptr);

Description

This service deletes a previously created Telnet Server instance.

Input Parameters

server_ptr   Pointer to Telnet Server control block.

Return Values

NX_SUCCESS   (0x00)   Successful Server delete.
NX_PTR_ERROR   (0x16)   Invalid Server pointer.
NX_CALLER_ERROR   (0x11)   Invalid caller of this service.

Allowed From

Threads

Example

```
/* Delete the Telnet Server instance "my_server".  */
status =  nx_telnet_server_delete(&my_server);

/* If status is NX_SUCCESS the Telnet Server was successfully deleted.  */
```

nx_telnet_server_disconnect

Disconnect a Telnet Client

Prototype

```
UINT nx_telnet_server_disconnect(NX_Telnet_SERVER *server_ptr,
                                     UINT logical_connection);
```

Description

This service disconnects a previously connected Client on this Telnet Server instance. This routine is typically called from the application's receive data callback function in response to a condition detected in the data received.

Input Parameters

server_ptr   Pointer to Telnet Server control block.

logical_connection Logical connection corresponding the Client connection on this Server. Valid value range from 0 through NX_TELENET_MAX_CLIENTS.

Return Values

NX_SUCCESS	(0x00)	Successful Server disconnect.
NX_OPTION_ERROR	(0x0A)	Invalid logical connection.
NX_PTR_ERROR	(0x16)	Invalid Server pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Disconnect the Telnet Client associated with logical connection 2 on
   the Telnet Server instance "my_server".  */
status =  nx_telnet_server_disconnect(&my_server, 2);

/* If status is NX_SUCCESS the Client on logical connection 2 was
   disconnected.  */
```

nx_telnet_server_get_open_connection_count

Return number of currently open connections

Prototype

```
UINT nx_telnet_server_get_open_connection_count(NX_TELNET_SERVER
              *server_ptr, UINT *connection_count);
```

Description

This service returns the number of currently connected Telnet Clients.

Input Parameters

server_ptr   Pointer to Telnet Server control block.

Connection_count
                      Pointer to memory to store connection count

Return Values

NX_SUCCESS	(0x00)	Successful completion.
NX_PTR_ERROR	(0x07)	Invalid Server pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Get the number of Telnet Clients connected to the Server. */

status = nx_telnet_server_get_open_connection_count(&my_server, &conn_count);

/* If status is NX_SUCCESS the conn_count holds the number of open connections.
*/
```

nx_telnet_server_packet_send

Send packet through Client connection

Prototype

```
UINT nx_telnet_server_packet_send(NX_Telnet_SERVER *server_ptr,
        UINT logical_connection, NX_PACKET *packet_ptr,
        ULONG wait_option);
```

Description

This service sends a packet to the Client connection on this Telnet Server instance. This routine is typically called from the application's receive data callback function in response to a condition detected in the data received.

Input Parameters

server_ptr   Pointer to Telnet Server control block.

logical_connection Logical connection corresponding the Client connection on this Server. Valid value range from 0 through NX_TELENET_MAX_CLIENTS.

packet_ptr   Pointer to the received packet.

wait_option   Defines how long the service will wait for the Telnet Server packet send. The wait options are defined as follows:

timeout value   (0x00000001 through 0xFFFFFFFE)
TX_WAIT_FOREVER (0xFFFFFFFF)

Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the Telnet Server responds to the request.

Selecting a numeric value (1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the Telnet Server response.

Return Values

NX_SUCCESS	(0x00)	Successful Server packet send.
NX_TELNET_FAILED	(0xF2)	Server socket send failed.
NX_OPTION_ERROR	(0x0A)	Invalid logical connection.
NX_PTR_ERROR	(0x16)	Invalid Server pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Send a packet to the Telnet Client associated with logical connection 2 on
   the Telnet Server instance "my_server".  */
status = nx_telnet_server_packet_send(&my_server, 2, my_packet, 100);

/* If status is NX_SUCCESS the packet was sent to the Client on logical
   connection 2.  */
```

nx_telnet_server_packet_pool_set

Set previously created packet pool as Telnet Server pool

Prototype

```
UINT nx_telnet_server_packet_pool_set(NX_TELNET_SERVER *server_ptr,
                                      NX_PACKET_POOL *packet_pool_ptr);
```

Description

This service sets a previously created packet pool as the Telnet Server packet pool if NX_TELNET_SERVER_USER_CREATE_PACKET_POOL is defined. It also requires that NX_TELNET_SERVER_OPTION_DISABLE not be defined such that the Telnet Server needs a packet pool to transmit Telnet options to Telnet clients.

This permits applications to create the packet pool in different memory e.g. no cache memory, than the Telnet Server stack. Note that if this function does not check if the Telnet Server packet pool is already set. If it is called on a non NULL Telnet Server packet pool pointer, it will overwrite it and replace the existing packet pool with packet pool pointed to by the input pointer.

Input Parameters

server_ptr   Pointer to Telnet Server control block

packet_pool_ptr   Pointer to previously created packet pool

Return Values

NX_SUCCESS   (0x00)   Successful Server
                                          packet send.
NX_PTR_ERROR   (0x07)   Invalid Server pointer.

Allowed From

Init, Threads

Example

```
status =  nx_packet_pool_create(&telnet_server_packet_pool, "Telnet Server Packet
                          Pool", telnet_server_pool_area, 600*10);

/* Set the packet pool as the Telnet Server packet pool.   */
status =  nx_telnet_server_packet_pool_set(&my_server,
                                      &telnet_server_packet_pool);

/* If status is NX_SUCCESS the packet pool is set as Telnet Server pool.  */
```

nx_telnet_server_start

Start a Telnet Server

Prototype

UINT **nx_telnet_server_start**(NX_Telnet_SERVER *server_ptr);

Description

This service starts a previously created Telnet Server instance.

Input Parameters

server_ptr   Pointer to Telnet Server control block.

Return Values

NX_SUCCESS   (0x00)   Successful Server start.
NX_TELNET_NO_PACKET_POOL
                        (0xF6)   No packet pool set
NX_PTR_ERROR   (0x16)   Invalid Server pointer.

Allowed From

Initialization, Threads

Example

```
/* Start the Telnet Server instance "my_server".  */
status =  nx_telnet_server_start(&my_server);

/* If status is NX_SUCCESS the Server was started.  */
```

nx_telnet_server_stop

Stop a Telnet Server

Prototype

UINT **nx_telnet_server_stop**(NX_Telnet_SERVER *server_ptr);

Description

This service stops a previously created and started Telnet Server instance.

Input Parameters

server_ptr   Pointer to Telnet Server control block.

Return Values

NX_SUCCESS	(0x00)	Successfully stopped
NX_PTR_ERROR	(0x16)	Invalid Server pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of service

Allowed From

Threads

Example

```
/* Stop the Telnet Server instance "my_server".  */
status =  nx_telnet_server_stop(&my_server);

/* If status is NX_SUCCESS the Server was stopped.  */
```