Telnet Protocol (Telnet) for NetX Duo

User Guide

Express Logic, Inc.

858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259

www.expresslogic.com

Part Number: 000-1055
Revision 5.2

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 NetXDuo 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 NetXDuo Telnet package has no further requirements.

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

NetX Duo Telnet is not changed in any way from NetX Telnet except when the Client attempts to connect to the server, the server host names must resolve to an NXD_ADDRESS, either IPv6 or IPv4 address. For backward compatibility, NetX Duo Telnet converted the original *nxd_telnet_client_connect* function to a wrapper function that will accept IPv4 addresses, convert them to NXD_ADDRESSES and pass a pointer to the data to the actual *nxd_telnet_client_connect* call. These will be discussed in greater detail in Chapter 3 and demonstrated in the “Small Example System” section in Chapter 2.

Telnet Constraints

The NetXDuo 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 *nxd_telnet.h* file.

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 NetXDuo 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 NetXDuo 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 NetXDuo Telnet Server calls the application specified callback function whenever a Telnet client ends the connection. It will also call the end connection callback when it detects an activity timeout on a currently open 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 Multi-Thread Support

The NetXDuo 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

NetXDuo 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 NetXDuo Telnet component.

Product Distribution

Telnet for NetX Duo 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:

nxd_telnet_client.h	Header file for Telnet Client for NetX Duo
nxd_telnet_client.c	C Source file for Telnet Client for NetX Duo
nxd_telnet_server.h	Header file for Telnet Server for NetX Duo
nxd_telnet_server.c	C Source file for Telnet Server for NetX Duo
nxd_telnet.pdf	PDF description of Telnet for NetX Duo
demo_netxduo_telnet.c	NetX DuoTelnet demonstration

Telnet Installation

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

Using Telnet

Using Telnet for NetXDuo is easy. Basically, the application code must include *nxd_telnet_server.h* for Telnet Server applications and *nxd_telnet_client.h* for Telnet Client applications after it includes *tx_api.h* and *nx_api.h*, in order to use ThreadX and NetX Duo. Oncethe header is included, the application code is then able to make the Telnet function calls specified later in this guide. The application must also include *nxd_telnet_client.c* and *nxd_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 NetXDuo Telnet.

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

Note also that because Telnet utilizes NetXDuo 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 NetXDuo Telnet is described in Figure 1.1 that appears below. In this example, the Telnet include files *are* brought in at line 7 and 8. Next, the Telnet Server is created in "*tx_application_define*" at line 146. Note that the Telnet Server and Client control blocks are defined as global variables at line 23-24 previously.

Before the Telnet Server or Client can be started they must validate their IP address with NetX Duo. For IPv4 connections this is accomplished by simply waiting briefly to let the NetX driver initialize the system on line 166. For IPv6 connections, this requires enabling IPv6 and ICMPv6 which it does in lines 171-172. The Client sets its global and linklocal IPv6 addresses on the primary interface on lines 181-186 and waits for NetX Duo validation to complete in the background. The Server also sets its global and linklocal addresses on its primary interface in lines 192 – 198. Note that the two services, *nxd_ipv6_global_address_set* and *nxd_ipv6_linklocal_address_set* are replaced with *nxd_ipv6_address_set service*. The former two services are still available for legacy NetX Duo applications but are eventually deprecated. Developers are encouraged to use *nxd_ipv6_address_set instead*.

After successful IP address validation with NetX Duo, the Telnet Server is started at line 215 using the *nxd_telnet_server_start* service. At line 226 the Telnet Client is created using the *nx_telnet_client_create* service. It then connects with the Telnet Server on line 242 for IPv4 applications and line 238 for IPv6 applications using the *nxd_telnet_client_connect* and *nx_telnet_client_connect* services respectively. After successful validation and connection with the server, it makes a few exchanges before disconnecting.

```
1    /* This is a small demo of TELNET on the high-performance NetX Duo TCP/IP stack.
2       This demo relies on ThreadX and NetX Duo to show a simple TELNET connection,
3       send, server echo, and then disconnection from the TELNET server.  */
4
5    #include  "tx_api.h"
6    #include  "nx_api.h"
7    #include  "nxd_telnet_client.h"
8    #include  "nxd_telnet_server.h"
9    #define     DEMO_STACK_SIZE         4096
10
11
12   /* Define the ThreadX and NetX object control blocks...  */
13
14   TX_THREAD               test_thread;
```

```
15  NX_PACKET_POOL          pool_server;
16  NX_PACKET_POOL          pool_client;
17  NX_IP                   ip_server;
18  NX_IP                   ip_client;
19
20
21  /* Define TELNET objects.  */
22
23  NX_TELNET_SERVER        my_server;
24  NX_TELNET_CLIENT        my_client;
25
26
27  #ifdef FEATURE_NX_IPV6
28
29  /* Define NetX Duo IP address for the NetX Duo Telnet Server and Client. */
30
31  NXD_ADDRESS     server_ip_address;
32  NXD_ADDRESS     client_ip_address;
33
34  #endif
35
36  #define         SERVER_ADDRESS          IP_ADDRESS(1,2,3,4)
37  #define         CLIENT_ADDRESS          IP_ADDRESS(1,2,3,5)
38
39
40
41  /* Define the counters used in the demo application...  */
42
43  ULONG                   error_counter;
44
45
46  /* Define timeout in ticks for connecting and sending/receiving data. */
47
48  #define                 TELNET_TIMEOUT  200
49
50  /* Define function prototypes.  */
51
52  void    thread_test_entry(ULONG thread_input);
53  void    _nx_ram_network_driver(struct NX_IP_DRIVER_STRUCT *driver_req);
54
55
56  /* Define the application's TELNET Server callback routines.  */
57
58  void    telnet_new_connection(NX_TELNET_SERVER *server_ptr, UINT
                                      logical_connection);
59  void    telnet_receive_data(NX_TELNET_SERVER *server_ptr, UINT logical_connection,
                                      NX_PACKET *packet_ptr);
60  void    telnet_connection_end(NX_TELNET_SERVER *server_ptr, UINT
                                      logical_connection);
61
62
63  /* Define main entry point.  */
64
65  intmain()
66  {
67
68      /* Enter the ThreadX kernel.  */
69      tx_kernel_enter();
70  }
71
72
73  /* Define what the initial system looks like.  */
74  void    tx_application_define(void *first_unused_memory)
75  {
76
77  UINT    status;
78  CHAR    *pointer;
79  UINT    iface_index, address_index;
80
81      /* Setup the working pointer.  */
82      pointer =  (CHAR *) first_unused_memory;
83
84      /* Create the main thread.  */
85      tx_thread_create(&test_thread, "test thread", thread_test_entry, 0,
86              pointer, DEMO_STACK_SIZE,
87              2, 2, TX_NO_TIME_SLICE, TX_AUTO_START);
88      pointer =  pointer + DEMO_STACK_SIZE;
89
90      /* Initialize the NetX system.  */
91      nx_system_initialize();
92
```

```
93      /* Create packet pool.  */
94      nx_packet_pool_create(&pool_server, "Server NetX Packet Pool",
                    600, pointer, 8192);
95      pointer = pointer + 8192;
96
97      /* Create an IP instance.  */
98      nx_ip_create(&ip_server, "Server NetX IP Instance", SERVER_ADDRESS,
99                           0xFFFFFF00UL, &pool_server, _nx_ram_network_driver,
100                          pointer, 4096, 1);
101
102     pointer =  pointer + 4096;
103
104     /* Create another packet pool. */
105     nx_packet_pool_create(&pool_client, "Client NetX Packet Pool", 600,
                                    pointer, 8192);
106     pointer = pointer + 8192;
107
108     /* Create another IP instance.  */
109     nx_ip_create(&ip_client, "Client NetX IP Instance", CLIENT_ADDRESS,
110                          0xFFFFFF00UL, &pool_client, _nx_ram_network_driver,
111                          pointer, 4096, 1);
112
113     pointer = pointer + 4096;
114
115     /* Enable ARP and supply ARP cache memory for IP Instance 0.  */
116     nx_arp_enable(&ip_server, (void *) pointer, 1024);
117     pointer = pointer + 1024;
118
119     /* Enable ARP and supply ARP cache memory for IP Instance 1.  */
120     nx_arp_enable(&ip_client, (void *) pointer, 1024);
121     pointer = pointer + 1024;
122
123     /* Enable TCP processing for both IP instances.  */
124     nx_tcp_enable(&ip_server);
125     nx_tcp_enable(&ip_client);
126
127
128 #ifdef FEATURE_NX_IPV6
129
130     /* Next set the NetX Duo Telnet Server and Client addresses. */
131     server_ip_address.nxd_ip_address.v6[3] = 0x105;
132     server_ip_address.nxd_ip_address.v6[2] = 0x0;
133     server_ip_address.nxd_ip_address.v6[1] = 0x0000f101;
134     server_ip_address.nxd_ip_address.v6[0] = 0x20010db1;
135     server_ip_address.nxd_ip_version = NX_IP_VERSION_V6;
136
137     client_ip_address.nxd_ip_address.v6[3] = 0x101;
138     client_ip_address.nxd_ip_address.v6[2] = 0x0;
139     client_ip_address.nxd_ip_address.v6[1] = 0x0000f101;
140     client_ip_address.nxd_ip_address.v6[0] = 0x20010db1;
141     client_ip_address.nxd_ip_version = NX_IP_VERSION_V6;
142
143 #endif
144
145     /* Create the NetX Duo TELNET Server.  */
146     status =  nx_telnet_server_create(&my_server, "Telnet Server", &ip_server,
147                     pointer, 2048, telnet_new_connection, telnet_receive_data,
148                     telnet_connection_end);
149
150     /* Check for errors.  */
151     if (status)
152         error_counter++;
153
154     return;
155 }
156
157
158 /* Define the test thread.  */
159 voidthread_test_entry(ULONG thread_input)
160 {
161
162 NX_PACKET   *my_packet;
163 UINT        status;
164
165     /* Allow other threads (e.g. IP thread task) to run first. */
166     tx_thread_sleep(100);
167
168 #ifdef FEATURE_NX_IPV6
169
170     /* Here's where we make the Telnet Client IPv6 enabled. */
171     nxd_ipv6_enable(&ip_client);
```

```
172      nxd_icmp_enable(&ip_client);
173
174      /* Wait till the IP task thread initializes the system. */
175      tx_thread_sleep(100);
176
177
178/* Set up the Client addresses on the Client IP for the primary interface. */
179      if_index = 0;
180
181status = nxd_ipv6_address_set(&ip_ client, iface_index, NX_NULL, 10,
                                   &address_index);
182status =nxd_ipv6_address_set(&ip_ client, iface_index, &client _ip_address,
                                        64, &address_index);
184
185      /* Allow NetX Duo time to validate addresses. */
186    tx_thread_sleep(400);
187
188
189/* Set up the Server addresses on the Client IP. */
190
191    iface_index = 0;
192status = nxd_ipv6_address_set(&ip_server, iface_index, NX_NULL, 10,
                                   &address_index);
193
194      status = nxd_ ipv6_address _set(&ip_server, iface_index, &server _ip_address,
                                        64, &address_index);

195
196
197      /* Allow NetX Duo time to validate addresses. */
198tx_thread_sleep(400);
199
200#endif
201
201
214      /* Start the TELNET Server.  */
215      status =  nx_telnet_server_start(&my_server);
216
217      /* Check for errors.  */
218      if (status != NX_SUCCESS)
219      {
220
221          return;
222      }
223
224
225      /* Create a TELENT client instance.  */
226      status =  nx_telnet_client_create(&my_client, "My TELNET Client",
                                                &ip_client, 600);
227
228      /* Check status.  */
229      if (status != NX_SUCCESS)
230      {
231          return;
232      }
233
234
235  #ifdef FEATURE_NX_IPV6
236
237      /* Connect the TELNET client to the TELNET Server at port 23.  */
238      status =  nxd_telnet_client_connect(&my_client, &server_ip_address, 23,
                                                TELNET_TIMEOUT);
239
240  #else
241      /* Connect the TELNET client to the TELNET Server at port 23.  */
242      status =  nx_telnet_client_connect(&my_client, SERVER_ADDRESS, 23,
                                             TELNET_TIMEOUT);
243  #endif
244
245      /* Check status.  */
246      if (status != NX_SUCCESS)
247      {
248          return;
249      }
250
251
252      /* Allocate a packet.  */
253      status =  nx_packet_allocate(&pool_client, &my_packet, NX_TCP_PACKET,
                                        NX_WAIT_FOREVER);
254
255      /* Check status.  */
```

```
256      if (status != NX_SUCCESS)
257      {
258          return;
259      }
260
261      /* Build a simple 1-byte message.  */
262      nx_packet_data_append(my_packet, "a", 1, &pool_client, NX_WAIT_FOREVER);
263
264      /* Send the packet to the TELNET Server.  */
265      status =  nx_telnet_client_packet_send(&my_client, my_packet, TELNET_TIMEOUT);
266
267      /* Check status.  */
268      if (status != NX_SUCCESS)
269      {
270          return;
271      }
272
273
274
275      /* Pickup the Server header.  */
276      status =  nx_telnet_client_packet_receive(&my_client, &my_packet,
                                                          TELNET_TIMEOUT);
277
278      /* Check status.  */
279      if (status != NX_SUCCESS)
280      {
281          return;
282      }
283
284
285      /* At this point the packet should contain the Server's banner
286         message sent by the Server callback function below.  Just
287         release it for this demo.  */
288      nx_packet_release(my_packet);
289
290      /* Pickup the Server echo of the character.  */
291      status =  nx_telnet_client_packet_receive(&my_client, &my_packet,
                                                    TELNET_TIMEOUT);
292
293      /* Check status.  */
294      if (status != NX_SUCCESS)
295      {
296          return;
297      }
298
299
300      /* At this point the packet should contain the character 'a' that
301         we sent earlier.  Just release the packet for now.  */
302      nx_packet_release(my_packet);
303
304      /* Now disconnect form the TELNET Server.  */
305      status =  nx_telnet_client_disconnect(&my_client, TELNET_TIMEOUT);
306
307
308      /* Check status.  */
309      if (status != NX_SUCCESS)
310      {
311          return;
312      }
313
314
315      /* Delete the TELNET Client.  */
316      status =  nx_telnet_client_delete(&my_client);
317
318      /* Check status.  */
319      if (status != NX_SUCCESS)
320      {
321          return;
322      }
323  }
324
325
326  /* This routine is called by the NetX Telnet Server whenever a new Telnet client
327     connection is established.  */
328  voidtelnet_new_connection(NX_TELNET_SERVER *server_ptr, UINT logical_connection)
329  {
330
331  UINT        status;
332  NX_PACKET   *packet_ptr;
333
334
```

```
335
336      /* Allocate a packet for client greeting. */
337      status =  nx_packet_allocate(&pool_server, &packet_ptr, NX_TCP_PACKET,
                                          NX_NO_WAIT);
338
339      if (status != NX_SUCCESS)
340      {
341          error_counter++;
342          return;
343      }
344
345
346      /* Build a banner message and a prompt.  */
347      nx_packet_data_append(packet_ptr,
               "**** Welcome to NetX TELNET Server ****\r\n\r\n\r\n", 45,&pool_server,
             NX_NO_WAIT);
348
349
350      nx_packet_data_append(packet_ptr, "NETX> ", 6, &pool_server, NX_NO_WAIT);
351
352      /* Send the packet to the client.  */
353      status =  nx_telnet_server_packet_send(server_ptr, logical_connection,
                      packet_ptr, TELNET_TIMEOUT);
354
355      if (status != NX_SUCCESS)
356      {
357          error_counter++;
358          nx_packet_release(packet_ptr);
359      }
360
361
362      return;
363  }
364
365
366  /* This routine is called by the NetX Telnet Server whenever data is present on a
367     Telnet client connection.  */
368  voidtelnet_receive_data(NX_TELNET_SERVER *server_ptr, UINT logical_connection,
                              NX_PACKET *packet_ptr)
369  {
370
371  UINT    status;
372  UCHAR   alpha;
373
374
375      /* This demo echoes the character back; on <cr,lf> sends a new prompt back to
376         the client.  A real system would likely buffer the character(s) received in a
377         buffer associated with the supplied logical connection and process it.  */
378
379
380      /* Just throw away carriage returns.  */
381      if ((packet_ptr ->nx_packet_prepend_ptr[0] == '\r') &&
               (packet_ptr ->nx_packet_length == 1))
382      {
383          printf("telnet server received just a CRLF\n");
384
385          nx_packet_release(packet_ptr);
386          return;
387      }
388
389
390      /* Setup new line on line feed.  */
391      if ((packet_ptr ->nx_packet_prepend_ptr[0] == '\n') || (packet_ptr ->
               nx_packet_prepend_ptr[1] == '\n'))
392      {
393
394          /* Clean up the packet.  */
395          packet_ptr ->nx_packet_length=  0;
396          packet_ptr ->nx_packet_prepend_ptr=  packet_ptr ->nx_packet_data_start +
                                                   NX_TCP_PACKET;
397          packet_ptr ->nx_packet_append_ptr =   packet_ptr ->nx_packet_data_start +
                                                   NX_TCP_PACKET;
398
399          /* Build the next prompt.  */
400          nx_packet_data_append(packet_ptr, "\r\nNETX> ", 8, &pool_server,
NX_NO_WAIT);
401
402          /* Send the packet to the client.  */
403          status =  nx_telnet_server_packet_send(server_ptr, logical_connection,
                                                   packet_ptr, TELNET_TIMEOUT);
404
```

```
405          if (status != NX_SUCCESS)
406          {
407              error_counter++;
408              nx_packet_release(packet_ptr);
409          }
410
411          return;
412      }
413
414
415      /* Pickup first character (usually only one from client).  */
416      alpha =  packet_ptr ->nx_packet_prepend_ptr[0];
417
418      /* Echo character.  */
419      status =  nx_telnet_server_packet_send(server_ptr, logical_connection,
                                                packet_ptr, TELNET_TIMEOUT);
420
421      if (status != NX_SUCCESS)
422      {
423          error_counter++;
424          nx_packet_release(packet_ptr);
425      }
426
427      /* Check for a disconnection.  */
428      if (alpha == 'q')
429      {
430
431          /* Initiate server disconnection.  */
432          nx_telnet_server_disconnect(server_ptr, logical_connection);
433      }
434  }
435
436
437  /* This routine is called by the NetX Telnet Server when the client disconnects.  */
438  voidtelnet_connection_end(NX_TELNET_SERVER *server_ptr, UINT logical_connection)
439  {
440      /* Cleanup any application specific connection or buffer information.  */
441      return;
442  }
```

Figure 1.1 Example of Telnet use with NetX Duo

Configuration Options

There are several configuration options for building Telnet for NetX Duo. 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. This define can be set by the application prior to inclusion of *nxd_telnet_server.h*.
NX_TELNET_SERVER_PRIORITY	The priority of the Telnet Server thread. By default, this value is defined as 16 to specify priority 16. This define can be set by the application prior to inclusion of *nxd_telnet_server.h*.
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. This define can be set by the application prior to inclusion of *nxd_telnet_server.h*and*nxd_telnet_client.h*
NX_TELNET_FRAGMENT_OPTION	Fragment enable for Telnet TCP requests. By default, this value is NX_DONT_FRAGMENT to disable Telnet TCP fragmenting. This define can be

set by the application prior to inclusion of *nxd_telnet_server.h* and*nxd_telnet_client.h*

NX_TELNET_SERVER_WINDOW_SIZE Server socket window size. By default, this value is 2048 bytes. This define can be set by the application prior to inclusion of *nxd_telnet_server.h*.

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, but can be redefined prior to inclusion of *nxd_telnet_server.h*and*nxd_telnet_client.h.*

NX_TELNET_SERVER_TIMEOUT Specifies the number of ThreadX ticks that internal services will suspend for. The default value is set to 1000, but can be redefined prior to inclusion of *nxd_telnet_server.h.*

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, but can be redefined prior to inclusion of *nxd_telnet_server.h.*

NX_TELNET_TIMEOUT_PERIOD Specifies the number of seconds between checking for Client activity timeouts. The default value is set to 60 seconds, but can be redefined prior to inclusion of *nxd_telnet_server.h.*

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 with IPv4 address

nxd_telnet_client_connect
Connect an IPv6 Telnet Client with IPv6 address

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_get_open_connection_count
Retrieve the number of open connections

nx_telnet_server_packet_send
Send packet through Client connection

nx_telnet_server_start
Start a Telnet Server

nx_telnet_server_stop
Stop a Telnet Server

nx_telnet_client_connect

Connect a Telnet Client with IPv4 address

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 using an IPv4 address for the Telnet Server. This service actually inserts the ULONG server IP address in an NXD_ADDRESS control block and sets the IP version to 4 before calling the *nxd_telnet_client_connect* service described below.

Input Parameters

client_ptr Pointer to Telnet Client control block.

server_ip IPv4 Address of the Telnet 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_ERROR	(0xF0)	Client connect error.
NX_TELNET_NOT_DISCONNECTED	(0xF4)	Client already connected.
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.  */
```

See Also

nx_telnet_client_create, nx_telnet_client_delete, nx_telnet_client_disconnect, nx_telnet_client_packet_receive, nx_telnet_client_packet_send, nx_telnet_server_create, nx_telnet_server_delete, nx_telnet_server_disconnect, nx_telnet_server_packet_send, nx_telnet_server_start, nx_telnet_server_stop

nxd_telnet_client_connect

Connect a Telnet Client with IPv6 or IPv4 address

Prototype

```
UINT nxd_telnet_client_connect(NX_TELNET_CLIENT *client_ptr,
            NXD_ADDRESS *server_ip_address, 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 using the Telnet Server's IPv6 address. This service can take an IPv4 or an IPv6 address but must be contained in the NXD_ADDRESS variable *server_ip_address.*

Input Parameters

client_ptr Pointer to Telnet Client control block.

server_ip_address 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_ERROR	(0xF0)	Client connect error.

NX_TELNET_NOT_DISCONNECTED (0xF4)		Client already connected.
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
   IPv6 address 20010db1:0:f101::101 and port 23.  */
status =  nxd_telnet_client_connect(&my_client, &server_ip_address, 23, 100);

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

See Also

nx_telnet_client_connect, nx_telnet_client_create, nx_telnet_client_delete, nx_telnet_client_disconnect, nx_telnet_client_packet_receive, nx_telnet_client_packet_send, nx_telnet_server_create, nx_telnet_server_delete, nx_telnet_server_disconnect, nx_telnet_server_packet_send, nx_telnet_server_start, nx_telnet_server_stop

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.
NX_TELNET_ERROR	(0xF0)	Client create error.
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.  */
```

See Also

nx_telnet_client_connect, nx_telnet_client_delete, nx_telnet_client_disconnect, nx_telnet_client_packet_receive, nx_telnet_client_packet_send, nx_telnet_server_create, nx_telnet_server_delete, nx_telnet_server_disconnect,

nx_telnet_server_packet_send, nx_telnet_server_start,
nx_telnet_server_stop

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.  */
```

See Also

nx_telnet_client_connect, nx_telnet_client_create, nx_telnet_client_disconnect, nx_telnet_client_packet_receive, nx_telnet_client_packet_send, nx_telnet_server_create, nx_telnet_server_delete, nx_telnet_server_disconnect, nx_telnet_server_packet_send, nx_telnet_server_start, nx_telnet_server_stop

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.  */
```

See Also

nx_telnet_client_connect, nx_telnet_client_create, nx_telnet_client_delete, nx_telnet_client_packet_receive, nx_telnet_client_packet_send, nx_telnet_server_create, nx_telnet_server_delete, nx_telnet_server_disconnect, nx_telnet_server_packet_send, nx_telnet_server_start, nx_telnet_server_stop

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.
NX_TELNET_ERROR	(0xF0)	Receive packet failed.
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.  */
```

See Also

nx_telnet_client_connect, nx_telnet_client_create, nx_telnet_client_delete, nx_telnet_client_disconnect, nx_telnet_client_packet_send, nx_telnet_server_create, nx_telnet_server_delete, nx_telnet_server_disconnect, nx_telnet_server_packet_send, nx_telnet_server_start, nx_telnet_server_stop

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 Client packet send.
NX_TELNET_ERROR	(0xF0)	Send packet failed – caller is responsible for releasing the packet.

NX_PTR_ERROR	(0x16)	Invalid Client or packet pointer.
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.  */
```

See Also

nx_telnet_client_connect, nx_telnet_client_create, nx_telnet_client_delete, nx_telnet_client_disconnect, nx_telnet_client_packet_receive, nx_telnet_server_create, nx_telnet_server_delete, nx_telnet_server_disconnect, nx_telnet_server_packet_send, nx_telnet_server_start, nx_telnet_server_stop

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 create.
NX_TELNET_ERROR	(0xF0)	Server create failed.
NX_PTR_ERROR	(0x16)	Invalid Server, IP, stack, or application callback pointers.

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.  */
```

See Also

nx_telnet_client_connect, nx_telnet_client_create, nx_telnet_client_delete, nx_telnet_client_disconnect, nx_telnet_client_packet_receive, nx_telnet_client_packet_send, nx_telnet_server_delete, nx_telnet_server_disconnect, nx_telnet_server_packet_send, nx_telnet_server_start, nx_telnet_server_stop

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_TELNET_ERROR	(0xF0)	Server delete failed.
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.  */
```

See Also

nx_telnet_client_connect, nx_telnet_client_create, nx_telnet_client_delete, nx_telnet_client_disconnect, nx_telnet_client_packet_receive, nx_telnet_client_packet_send, nx_telnet_server_create, nx_telnet_server_disconnect, nx_telnet_server_packet_send, nx_telnet_server_start, nx_telnet_server_stop

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_TELNET_ERROR	(0xF0)	Server disconnect 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

```
/* 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.  */
```

See Also

nx_telnet_client_connect, nx_telnet_client_create, nx_telnet_client_delete, nx_telnet_client_disconnect, nx_telnet_client_packet_receive, nx_telnet_client_packet_send, nx_telnet_server_create, nx_telnet_server_delete, nx_telnet_server_packet_send, nx_telnet_server_start, nx_telnet_server_stop

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	(0x16)	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.
*/
```

See Also

nx_telnet_client_connect, nx_telnet_client_create, nx_telnet_client_delete, nx_telnet_client_disconnect, nx_telnet_client_packet_receive, nx_telnet_client_packet_send, nx_telnet_server_create, nx_telnet_server_delete, nx_telnet_server_disconnect, nx_telnet_server_start, nx_telnet_server_stop

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 packet 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.  */
```

See Also

nx_telnet_client_connect, nx_telnet_client_create, nx_telnet_client_delete, nx_telnet_client_disconnect, nx_telnet_client_packet_receive, nx_telnet_client_packet_send, nx_telnet_server_create, nx_telnet_server_delete, nx_telnet_server_disconnect, nx_telnet_server_start, nx_telnet_server_stop

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_ERROR	(0xF0)	Server start failed.
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.  */
```

See Also

nx_telnet_client_connect, nx_telnet_client_create, nx_telnet_client_delete, nx_telnet_client_disconnect, nx_telnet_client_packet_receive, nx_telnet_client_packet_send, nx_telnet_server_create, nx_telnet_server_delete, nx_telnet_server_disconnect, nx_telnet_server_packet_send, nx_telnet_server_stop

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)	Successful Server stop.
NX_TELNET_ERROR	(0xF0)	Server stop failed.
NX_PTR_ERROR	(0x16)	Invalid Server pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this 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.  */
```

See Also

nx_telnet_client_connect, nx_telnet_client_create, nx_telnet_client_delete, nx_telnet_client_disconnect, nx_telnet_client_packet_receive, nx_telnet_client_packet_send, nx_telnet_server_create, nx_telnet_server_delete, nx_telnet_server_disconnect, nx_telnet_server_packet_send, nx_telnet_server_start