File Transfer Protocol (NetX Duo FTP)

User Guide

Express Logic, Inc.

858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259

www.expresslogic.com

Part Number: 000-1052
Revision 5.2

Contents

Chapter 1

Introduction to NetX Duo FTP

The File Transfer Protocol (FTP) is a protocol designed for file transfers. FTP utilizes reliable Transmission Control Protocol (TCP) services to perform its file transfer function. Because of this, FTP is a highly reliable file transfer protocol. FTP is also high-performance. The actual FTP file transfer is performed on a dedicated FTP connection. NetX Duo FTP accommodates both IPv4 and IPv6 networks. IPv6 does not directly change the FTP protocol, although some changes in the original NetX FTP API are necessary to accommodate IPv6 and will be described in this document.

FTP Requirements

In order to function properly, the NetX FTP package requires NetX Duo. The host application must create an IP instance for running NetX services and periodic tasks. If running the FTP host application over an IPv6 network, IPv6, and ICMPv6 must be enabled on the IP task. TCP must be also enabled for either IPv6 or IPv4 networks. The IPv6 host application must set its linklocal and global IPv6 address using the IPv6 API and/or DHCPv6. A demo program in section “Small Example System” in **Chapter 2** demonstrates how this is done.

The FTP Server and Client are also designed to work with the FileX embedded file system. If FileX is not available, the host developer can implement or substitute their own file system along the guidelines suggested in filex_stub.h by defining each of the services listed in that file. This is discussed in later sections of this guide.

The FTP Client portion of the NetX FTP package has no further requirements.

The FTP Server portion of the NetX FTP package has several additional requirements. First, it requires complete access to TCP *well-known port 21* for handling all Client FTP command requests and *well-known port 20* for handling all Client FTP data transfers.

FTP Constraints

The FTP standard has many options regarding the representation of file

data. NetX FTP does not implement switch options e.g. ls –al. NetX FTP Server expects to receive requests and their arguments in a single packet rather than consecutive packets.

Similar to UNIX implementations, NetX FTP assumes the following file format constraints:

File Type:	**Binary**
File Format:	**Nonprint Only**
File Structure:	**File Structure Only**

FTP File Names

FTP file names should be in the format of the target file system (usually FileX). They should be NULL terminated ASCII strings, with full path information if necessary. There is no specified limit for the size of FTP file names in the NetX FTP implementation. However, the packet pool payload size should be able to accommodate the maximum path and/or file name.

FTP Client Commands

The FTP has a simple mechanism for opening connections and performing file and directory operations. There is basically a set of standard FTP commands that are issued by the Client after a connection has been successfully established on the TCP *well-known port 21*. The following shows some of the basic FTP commands. Note that the only difference when FTP runs over IPv6 is that the PORT command is replaced with the EPRT command:

FTP Command	**Meaning**
CWD path	*Change working directory*
DELE filename	*Delete specified file name*
EPRT ip_address, port	*Provide IPv6 address and Client data port*
LIST directory	*Get directory listing*
MKD directory	*Make new directory*
NLST directory	*Get directory listing*
NOOP	*No operation, returns success*
PASS password	*Provide password for login*
PORT ip_address,port	*Provide IP address and Client data port*
PWD path	*Pickup current directory path*
QUIT	*Terminate Client connection*
RETR filename	*Read specified file*
RMD directory	*Delete specified directory*
RNFR oldfilename	*Specify file to rename*
RNTO newfilename	*Rename file to supplied file name*
STOR filename	*Write specified file*
TYPE I	*Select binary file image*
USER username	*Provide username for login*

These ASCII commands are used internally by the NetX FTP Client software to perform FTP operations with the FTP Server.

FTP Server Responses

Once the FTP Server processes the Client request, it returns a 3-digit coded response in ASCII followed by optional ASCII text. The numeric response is used by the FTP Client software to determine whether the operation succeeded or failed. The following list shows various FTP Server responses to Client requests:

First Numeric Field	**Meaning**
1xx	*Positive preliminary status – another reply coming*.
2xx	*Positive completion status*.
3xx	*Positive preliminary status – another command must be sent*.
4xx	*Temporary error condition*.
5xx	*Error condition.*

Second Numeric Field	**Meaning**
x0x	Syntax error in command.
x1x	Informational message.
x2x	Connection related.
x3x	Authentication related.
x4x	Unspecified.
x5x	File system related.

For example, a Client request to disconnect an FTP connection with the QUIT command will typically be responded with a “221” code from the Server – if the disconnect is successful.

FTP Communication

The FTP Server utilizes the *well-known TCP port 21* to field Client requests. FTP Clients may use any available TCP port. The general sequence of FTP events is as follows. As mentioned previous, the only difference with FTP running over IPv6 is the PORT command is replaced with the EPRT command:

FTP Read File Requests:

1. Client issues TCP connect to Server port 21.
2. Server sends “220” response to signal success.
3. Client sends “USER” message with “username.”
4. Server sends “331” response to signal success.
5. Client sends “PASS” message with “password.”
6. Server sends “230” response to signal success.

7. Client sends “TYPE I” message for binary transfer.
8. Server sends “200” response to signal success.
9. Client sends “PORT” message with IP address and port.
10. Server sends “200” response to signal success.
11. Client sends “RETR” message with file name to read.
12. Server creates data socket and connects with client data port specified in the “EPRT” command.
13. Server sends “125” response to signal file read has started.
14. Server sends contents of file through the data connection. This process continues until file is completely transferred.
15. When finished, Server disconnects data connection.
16. Server sends “250” response to signal file read is successful.
17. Clients sends “QUIT” to terminate FTP connection.
18. Server sends “221” response to signal disconnect is successful.
19. Server disconnects FTP connection.

FTP Write Requests:

1. Client issues TCP connect to Server port 21.
2. Server sends “220” response to signal success.
3. Client sends “USER” message with “username.”
4. Server sends “331” response to signal success.
5. Client sends “PASS” message with “password.”
6. Server sends “230” response to signal success.
7. Client sends “TYPE I” message for binary transfer.
8. Server sends “200” response to signal success.
9. *IPv6 applications*: Client sends “EPRT” message with IP address and port.
   IPv4 applications: Client sends “PORT” message with IP address and port.
10. Server sends “200” response to signal success.
11. Client sends “STOR” message with file name to write.
12. Server creates data socket and connects with client data port specified in the previous “EPRT” or “PORT” command.
13. Server sends “125” response to signal file write has started.
14. Client sends contents of file through the data connection. This process continues until file is completely transferred.
15. When finished, Client disconnects data connection.
16. Server sends “250” response to signal file write is successful.
17. Clients sends “QUIT” to terminate FTP connection.
18. Server sends “221” response to signal disconnect is successful.
19. Server disconnects FTP connection.

FTP Authentication

Whenever an FTP connection takes place, the Client must provide the Server with a *username* and *password*. Some FTP sites allow what is called *Anonymous FTP*, which allows FTP access without a specific username and password. For this type of connection, “anonymous” should be supplied for username and the password should be a complete e-mail address.

The user is responsible for supplying NetX FTP with login and logout authentication routines. These are supplied during the ***nxd_ftp_server_create*** and ***nx_ftp_server_create*** services and called from the password processing. The difference between the two is the ***nxd_ftp_server_create*** input function pointers to login and logout authenticate functions expect the NetX Duo address type ***NXD_ADDRESS***. This data type holds both IPv4 or IPv6 address formats, making this function the “duo” service supporting both IPv4 and IPv6 networks. The ***nx_ftp_server_create*** input function pointers to login and logout authenticate functions expect ULONG IP address type. This function is limited to IPv4 networks. The developer is encouraged to use the “duo” service whenever possible.

If the *login* function returns NX_SUCCESS, the connection is authenticated and FTP operations are allowed. Otherwise, if the *login* function returns something other than NX_SUCCESS, the connection attempt is rejected.

FTP Multi-Thread Support

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

FTP RFCs

NetX Duo FTP is compliant with RFC 959, RFC 2428 and related RFCs.

Chapter 2

Installation and Use of FTP

This chapter contains a description of various issues related to installation, set up, and usage of the NetX Duo FTP services.

Product Distribution

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

nxd_ftp_client.h	Header file for NetX Duo FTP Client
nxd_ftp_client.c	C Source file for NetX Duo FTP Client
nxd_ftp_server.h	Header file for NetX Duo FTP Server
nxd_ftp_server.c	C Source file for NetX Duo FTP Server
filex_stub.h	Stub file if FileX is not present
nxd_ftp.pdf	PDF description of FTP for NetX Duo
demo_netxduo_ftp.c	FTP demonstration system

NetX Duo FTP Installation

In order to use the NetX Duo FTP API, 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_ftp_client.h* and *nxd_ftp_client.c* should be copied into this directory for FTP Client applications, and *nxd_ftp_server.h* and *nxd_ftp_server.c* files should be copied into this directory for FTP Server applications.

Using NetX Duo FTP

Using the NetX Duo FTP API is easy. Basically, the application code must include either *nxd_ftp_client.h* for FTP Client applications or *nxd_ftp_server* for FTP Server applications, after it includes *tx_api.h, fx_api.h,* and *nx_api.h*, in order to use ThreadX, FileX, and NetX Duo, respectively. The build project must include the FTP source code and the host application file, and of course the ThreadX and NetX library files. This is all that is required to use NetX Duo FTP.

Note that since FTP utilizes NetX Duo TCP services, TCP must be enabled with the *nx_tcp_enable* call prior to using FTP.

Note that the NetX Duo library can be enabled for IPv6 and still support IPv4 networks. However, NetX Duo cannot support IPv6 unless it is enabled. To disable IPv6 processing in NetX Duo, the **NX_DISABLE_IPV6** must be defined in the *nx_user.h* file, and that file must be included in the NetX Duo library build by defining **NX_INCLUDE_USER_DEFINE_FILE** in the *nx_port.h* file. By default, **NX_DISABLE_IPV6** is not defined (IPv6 is enabled). This is different from the *nxd_ipv6_enable* service that sets up the IPv6 protocols and services on the IP task, and requires **NX_DISABLE_IPV6** to be not defined.

Small Example System of NetX Duo FTP

An example of how easy it is to use NetX Duo FTP is described in Figure 1.1 that appears below. In this example, both an FTP Server and an FTP Client are created. Therefore both FTP include files *nxd_ftp_client.h* and *nxd_ftp_server.h are* brought in at line 8 and 9. Next, the FTP Server is created in "*tx_application_define*" at line 86. Note that the FTP Server and Client control blocks are defined as a global variables at line 26 previously.

This demo shows how to use the duo functions available in NetX Duo FTP as well as the legacy IPv4 limited FTP services. To use the duo functions, the demo defines FTP_DUO (line

At line 145 the FTP Server is created with ***nxd_ftp_server_create*** if the host application defines FTP_DUO which supports both IPv4 and IPv6. If it is not, the FTP Server is created with ***nx_ftp_server_create*** on line 148 with the IPv4 limited service. Note that the 'duo' function uses different login and logout function arguments than the IPv4 service, both of which are defined at the bottom of the file on lines 409 -441.

The FTP server must then establish its IPv6 address (global and link local) with NetX Duo, starting at line 370 in the FTP server thread entry function. The FTP server is then started on line 395 and is ready for FTP client requests.

The FTP Client is created in line 251 and goes through the same process as the FTP Server to get the FTP Client IP task IPv6 enabled, and its IPv6 addresses validated starting at line 226.

Then the Client connects to the FTP Server using ***nxd_ftp_client_connect*** in line 271 if it has defined FTP_DUO, or line 277 if it is using the IPv4 limited service ***nx_ftp_client_connect***. Over the course of the FTP Client thread function, it writes a file to the FTP server and reads it back before disconnecting.

```
1    /* This is a small demo of NetX Duo FTP on the high-performance NetX Duo
```

```
2       TCP/IP stack.  This demo relies on ThreadX, NetX Duo, and FileX to show a 3
        simple file transfer from the client and then back to the server.  */
4
5     #include    "tx_api.h"
6     #include    "fx_api.h"
7     #include    "nx_api.h"
8     #include    "nxd_ftp_server.h"
9     #include    “nxd_ftp_client.h”
10    #define     DEMO_STACK_SIZE         4096
11
12    /* Define the ThreadX, NetX, and FileX object control blocks...  */
13
14    TX_THREAD               server_thread;
15    TX_THREAD               client_thread;
16    NX_PACKET_POOL          server_pool;
17    NX_IP                   server_ip;
18    NX_PACKET_POOL          client_pool;
19    NX_IP                   client_ip;
20    FX_MEDIA                ram_disk;
21
22
23    /* Define the NetX FTP object control blocks.  */
24
25    NX_FTP_CLIENT           ftp_client;
26    NX_FTP_SERVER           ftp_server;
27
28
29    /* Define the counters used in the demo application...  */
30
31    ULONG                   error_counter;
32
33
34    /* Define the memory area for the FileX RAM disk.  */
35
36    UCHAR                   ram_disk_memory[32000];
37    UCHAR                   ram_disk_sector_cache[512];
38
39
40    #define FTP_SERVER_ADDRESS  IP_ADDRESS(1,2,3,4)
41    #define FTP_CLIENT_ADDRESS  IP_ADDRESS(1,2,3,5)
42
43
44    /* Define the FileX and NetX driver entry functions.  */
45    VOID    _fx_ram_driver(FX_MEDIA *media_ptr);
46    VOID    _nx_ram_network_driver(NX_IP_DRIVER *driver_req_ptr);
47    void    client_thread_entry(ULONG thread_input);
48    void    thread_server_entry(ULONG thread_input);
49
50    #define FTP_DUO   /* Use the NetX Duo services for IPv6 support */
51
52
53    #ifdef FTP_DUO
54
55    /* Define NetX Duo IP address for the NetX Duo FTP Server and Client. */
56    NXD_ADDRESS     server_ip_address;
57    NXD_ADDRESS     client_ip_address;
58
59    #endif
60
61
62    /* Define server login/logout functions.  These are stubs for functions that
63       would validate a client login request.   */
64
65    #ifdef FTP_DUO
66    UINT    server_login_duo(struct NX_FTP_SERVER_STRUCT *ftp_server_ptr, NXD_ADDRESS
              *client_ipduo_address, UINT client_port, CHAR *name, CHAR
              *password, CHAR *extra_info);
67    UINT    server_logout_duo(struct NX_FTP_SERVER_STRUCT *ftp_server_ptr,
              NXD_ADDRESS *client_ipduo_address, UINT client_port, CHAR *name, CHAR
              *password, CHAR *extra_info);
68    #else
69    UINT    server_login(struct NX_FTP_SERVER_STRUCT *ftp_server_ptr, ULONG
              client_ip_address, UINT client_port, CHAR *name, CHAR *password,
              CHAR *extra_info);
70    UINT    server_logout(struct NX_FTP_SERVER_STRUCT *ftp_server_ptr, ULONG
              client_ip_address, UINT client_port, CHAR *name, CHAR *password, CHAR
              *extra_info);
71    #endif
72
73
74    /* Define main entry point.  */
```

```
75
76   int main()
77   {
78
79       /* Enter the ThreadX kernel.  */
80       tx_kernel_enter();
81   }
82
83
84   /* Define what the initial system looks like.  */
85
86   void    tx_application_define(void *first_unused_memory)
87   {
88
89   UINT    status;
90   UCHAR   *pointer;
91
92
93       /* Setup the working pointer.  */
94       pointer =  (UCHAR *) first_unused_memory;
95
96       /* Create a helper thread for the server. */
97       tx_thread_create(&server_thread, "FTP Server thread",
                         thread_server_entry, 0,
98                       pointer, DEMO_STACK_SIZE,
99                       1, 1, TX_NO_TIME_SLICE, TX_AUTO_START);
100
101      pointer =  pointer + DEMO_STACK_SIZE;
102
103      /* Initialize NetX.  */
104      nx_system_initialize();
105
106      /* Create the packet pool for the FTP Server.  */
107      status = nx_packet_pool_create(&server_pool, "NetX Server Packet Pool",
                                        256, pointer, 8192);
108      pointer = pointer + 8192;
109
110      /* Check for errors.  */
111      if (status)
112          error_counter++;
113
114      /* Create the IP instance for the FTP Server.  */
115      status = nx_ip_create(&server_ip, "NetX Server IP Instance",
                   FTP_SERVER_ADDRESS, 0xFFFFFF00UL, &server_pool,
                   _nx_ram_network_driver, pointer, 2048, 1);
116
117      pointer = pointer + 2048;
118
119      /* Check for errors.  */
120      if (status)
121          error_counter++;
122
123      /* Enable ARP and supply ARP cache memory for server IP instance.  */
124      nx_arp_enable(&server_ip, (void *) pointer, 1024);
125      pointer = pointer + 1024;
126
127      /* Enable TCP.  */
128      nx_tcp_enable(&server_ip);
129
130  #ifdef FTP_DUO
131      /* Create the FTP server using the duo service. */
132      /* Set the NetX Duo FTP Server and Client addresses. */
133      server_ip_address.nxd_ip_address.v6[3] = 0x105;
134      server_ip_address.nxd_ip_address.v6[2] = 0x0;
135      server_ip_address.nxd_ip_address.v6[1] = 0x0000f101;
136      server_ip_address.nxd_ip_address.v6[0] = 0x20010db1;
137      server_ip_address.nxd_ip_version = NX_IP_VERSION_V6;
138
139      client_ip_address.nxd_ip_address.v6[3] = 0x101;
140      client_ip_address.nxd_ip_address.v6[2] = 0x0;
141      client_ip_address.nxd_ip_address.v6[1] = 0x0000f101;
142      client_ip_address.nxd_ip_address.v6[0] = 0x20010db1;
143      client_ip_address.nxd_ip_version = NX_IP_VERSION_V6;
144
145      status =  nxd_ftp_server_create(&ftp_server, "FTP Server Instance", &server_ip,
                                         &ram_disk, pointer, DEMO_STACK_SIZE,
                                        &server_pool, server_login_duo, server_logout);
146  #else
147      /* Create the server using IPv4 only service. */
148      status =  nx_ftp_server_create(&ftp_server, "FTP Server Instance", &server_ip,
```

```
                                    &ram_disk, pointer, DEMO_STACK_SIZE,
                                   &server_pool, server_login, server_logout);
149 #endif
150
151      pointer =  pointer + DEMO_STACK_SIZE;
152
153
154      /* Now set up the FTP Client. */
155      /* Create the main FTP client thread.  */
156      status = tx_thread_create(&client_thread, "FTP Client thread ",
               client_thread_entry, 0,
157              pointer, DEMO_STACK_SIZE,
158              2, 2, TX_NO_TIME_SLICE, TX_AUTO_START);
159
160      pointer += DEMO_STACK_SIZE ;
161
162      /* Create a packet pool for the FTP client.  */
163      status =  nx_packet_pool_create(&client_pool, "NetX Client Packet Pool",
                                         256, pointer, 8192);
164
165      pointer =  pointer + 8192;
166
167      /* Create IP instance for the FTP client regardless if IPv6 or IPv4 network.  */
168      status = nx_ip_create(&client_ip, "NetX Client IP Instance",
                               FTP_CLIENT_ADDRESS, 0xFFFFFF00UL,
                               &client_pool, _nx_ram_network_driver, pointer, 2048,
                               1);
170      pointer = pointer + 2048;
171
172      /* Enable ARP and supply ARP cache memory for the FTP Client IP.  */
173      nx_arp_enable(&client_ip, (void *) pointer, 1024);
174
175      pointer = pointer + 1024;
176
177      /* Enable TCP for client IP instance.  */
178      nx_tcp_enable(&client_ip);
179
180      return;
181
182  }
183
184  /* Define the FTP client thread.  */
185
186  void    client_thread_entry(ULONG thread_input)
187  {
188
189  NX_PACKET   *my_packet;
190  UINT        status;
191  UINT        old_priority;
192
193
194      /* Format the RAM disk - the memory for the RAM disk was defined above.  */
195      status = fx_media_format(&ram_disk,
196                               _fx_ram_driver,               // Driver entry
197                               ram_disk_memory,              // RAM disk
                                                               // memory pointer
198                               ram_disk_sector_cache,        // Media buffer
                                                               // pointer
199                               sizeof(ram_disk_sector_cache), // Media buffer
                                                               // size
200                               "MY_RAM_DISK",                // Volume Name
201                               1,                            // Number of FATs
202                               32,                           // Directory
                                                               // Entries
203                               0,                            // Hidden sectors
204                               256,                          // Total sectors
205                               128,                          // Sector size
206                               1,                            // Sectors per
                                                               // cluster
207                               1,                            // Heads
208                               1);                           // Sectors per
                                                               // track
209
210      /* Check status.  */
211      if (status != NX_SUCCESS)
212          error_counter++;
213
214      /* Open the RAM disk.  */
215      status = fx_media_open(&ram_disk, "RAM DISK", _fx_ram_driver,
               ram_disk_memory, ram_disk_sector_cache, sizeof(ram_disk_sector_cache));
216
```

```
217      /* Check status.  */
218      if (status != NX_SUCCESS)
219          error_counter++;
220
221      /* Let the IP threads execute.    */
222      tx_thread_relinquish();
223
224  #ifdef FTP_DUO
225
226      /* Here's where we make the FTP Client IP task IPv6 enabled. */
227      nxd_ipv6_enable(&client_ip);
228      nxd_icmp_enable(&client_ip);
229
230      /* Wait till the IP task thread has had a chance to set the device MAC
           address. */
231      while (client_ip.nx_ip_arp_physical_address_msw == 0 ||
232             client_ip.nx_ip_arp_physical_address_lsw == 0)
233      {
234
235          tx_thread_sleep(50);
236      }
237
238      nxd_ipv6_linklocal_address_set(&client_ip, NULL);
239
240      nxd_ipv6_global_address_set(&client_ip, &client_ip_address, 64);
241
242       /* Allow NetX Duo time to validate addresses. */
243
244      tx_thread_sleep(400);
245
246
247  #endif
248
249
250      /* Create an FTP client.  */
251      status =  nx_ftp_client_create(&ftp_client, "FTP Client", &client_ip,
                                       2000, &client_pool);
252
253      /* Check status.  */
254      if (status != NX_SUCCESS)
255          error_counter++;
256
257
258  #ifdef FTP_DUO
259
260
261
262      /* Now connect with the NetX FTP server over IPv6 using the 'duo' service.
263         The server_ip_address variable can hold an IPv4 address so this works
264         With either IPv4 or IPv6 networks. */
267      do
268      {
269
270          /* Now connect with the NetX Duo FTP server. */
271          status =  nxd_ftp_client_connect(&ftp_client, &server_ip_address,
                                             "name", "password", 100);
272      } while (status != NX_SUCCESS);
273
274  #else
275
276      /* Now connect with the NetX FTP (IPv4) server. */
277      status =  nx_ftp_client_connect(&ftp_client, FTP_SERVER_ADDRESS, "name",
                                        "password", 100);
278
279  #endif
280
281      /* Check status.  */
282      if (status != NX_SUCCESS)
283          error_counter++;
284
285
286      /* Open a FTP file for writing.  */
287      status =  nx_ftp_client_file_open(&ftp_client, "test.txt",
                                          NX_FTP_OPEN_FOR_WRITE, 100);
288
289      /* Check status.  */
290      if (status != NX_SUCCESS)
291          error_counter++;
292
293      /* Allocate a FTP packet.  */
294      status =  nx_packet_allocate(&client_pool, &my_packet, NX_TCP_PACKET,100);
```

```
295
296     /* Check status.  */
297     if (status != NX_SUCCESS)
298         error_counter++;
299
300     /* Write ABCs into the packet payload!  */
301     memcpy(my_packet -> nx_packet_prepend_ptr, "ABCDEFGHIJKLMNOPQRSTUVWXYZ  ", 28);
302
303     /* Adjust the write pointer.  */
304     my_packet -> nx_packet_length =  28;
305     my_packet -> nx_packet_append_ptr =  my_packet -> nx_packet_prepend_ptr + 28;
306
307     /* Write the packet to the file test.txt.  */
308     status =  nx_ftp_client_file_write(&ftp_client, my_packet, 100);
309
310     /* Check status.  */
311     if (status != NX_SUCCESS)
312         error_counter++;
313
314
315     /* Close the file.  */
316     status =  nx_ftp_client_file_close(&ftp_client, 100);
317
318     /* Check status.  */
319     if (status != NX_SUCCESS)
320         error_counter++;
321
322
323     /* Now open the same file for reading.  */
324     status =  nx_ftp_client_file_open(&ftp_client, "test.txt",
                                          NX_FTP_OPEN_FOR_READ, 100);
325
326     /* Check status.  */
327     if (status != NX_SUCCESS)
328         error_counter++;
329
330     /* Read the file.  */
331     status =  nx_ftp_client_file_read(&ftp_client, &my_packet, 100);
332
333     /* Check status.  */
334     if (status != NX_SUCCESS)
335         error_counter++;
336     else
337         nx_packet_release(my_packet);
338
339     /* Close this file.  */
340     status =  nx_ftp_client_file_close(&ftp_client, 100);
341
342     if (status != NX_SUCCESS)
343         error_counter++;
344
345     /* Disconnect from the server.  */
346     status =  nx_ftp_client_disconnect(&ftp_client, 100);
347
348     /* Check status.  */
349     if (status != NX_SUCCESS)
350         error_counter++;
351
352
353     /* Delete the FTP client.  */
354     status =  nx_ftp_client_delete(&ftp_client);
355
356     /* Check status.  */
357     if (status != NX_SUCCESS)
358         error_counter++;
359 }
360
361
362 /* Define the helper FTP server thread.  */
363 void    thread_server_entry(ULONG thread_input)
364 {
365
366 UINT            status;
367
368
369 #ifdef FTP_DUO
370
371     /* Here's where we make the FTP server IPv6 enabled. */
372     nxd_ipv6_enable(&server_ip);
373     nxd_icmp_enable(&server_ip);
374
```

```
375     /* Wait till the IP task thread has had a chance to set the device MAC
           address. */
376     while (server_ip.nx_ip_arp_physical_address_msw == 0 ||
377             server_ip.nx_ip_arp_physical_address_lsw == 0)
378     {
379         tx_thread_sleep(30);
380     }
381
382     nxd_ipv6_linklocal_address_set(&server_ip, NULL);
383     nxd_ipv6_global_address_set(&server_ip, &server_ip_address, 64);
384
385     /* Wait for NetX Duo to validate server address. */
386     while (server_ip.nx_ipv6_global.nxd_state != NX_IPV6_ADDR_STATE_VALID)
387     {
388         tx_thread_sleep(100);
389     }
390
391 #endif
392
393
394     /* OK to start the FTP Server.   */
395     status = nx_ftp_server_start(&ftp_server);
396
397     if (status != NX_SUCCESS)
398         error_counter++;
399
400     /* FTP server ready to take requests! */
401
402     /* Let the IP threads execute.    */
403     tx_thread_relinquish();
404
405     return;
406 }
407
408
409 #ifdef FTP_DUO

      /* IPv6 or IPv6 login and logout authentication callbacks. */
410 UINT  server_login_duo(struct NX_FTP_SERVER_STRUCT *ftp_server_ptr, NXD_ADDRESS
                    *client_ipduo_address, UINT client_port,
411                  CHAR *name, CHAR *password, CHAR *extra_info)
412 {
413
414     /* Always return success.  */
415     return(NX_SUCCESS);
416 }
417
418 UINT  server_logout_duo(struct NX_FTP_SERVER_STRUCT *ftp_server_ptr, NXD_ADDRESS
                     *client_ipduo_address, UINT client_port,
419                   CHAR *name, CHAR *password, CHAR *extra_info)
420 {
421
422     /* Always return success.  */
423     return(NX_SUCCESS);
424 }
425 #else
426
427 /* IPv4 login and logout authentication callbacks. */
428 UINT  server_login(struct NX_FTP_SERVER_STRUCT *ftp_server_ptr, ULONG
                         client_ip_address, UINT client_port, CHAR *name, CHAR
                         *password, CHAR *extra_info)
429 {
430
431     /* Always return success.  */
432     return(NX_SUCCESS);
433 }
434
435 UINT  server_logout(struct NX_FTP_SERVER_STRUCT *ftp_server_ptr, ULONG
                         client_ip_address, UINT client_port, CHAR *name, CHAR
                         *password, CHAR *extra_info)
436 {
437
438     /* Always return success.  */
439     return(NX_SUCCESS);
440 }
441 #endif
```

Figure 1.1 Example of NetX Duo FTP

Configuration Options

There are several configuration options for building NetX FTP and NetX Duo FTP. The default values are listed, but each define can be set by the application prior to inclusion of the specified NetX Duo FTP header file. If no header file is specified, the option is available in both *nxd_ftp_client.h* and *nxd_ftp_server.h*. The following list describes each in detail:

Define	Meaning
NX_FTP_SERVER_PRIORITY	The priority of the FTP Server thread. By default, this value is defined as 16 to specify priority 16 in *nxd_ftp_server.h.*
NX_FTP_MAX_CLIENTS	The maximum number of Clients the Server can handle at one time. By default, this value is 4 to support 4 Clients at once in *nxd_ftp_server.h.*
NX_FTP_SERVER_TIMEOUT	Specifies the number of ThreadX ticks that internal services will suspend for. The default value is set to 100 in *nxd_ftp_server.h.*
NX_FTP_ACTIVITY_TIMEOUT	Specifies the number of seconds a Client connection is maintained if there is no activity. The default value is set to 240 in *nxd_ftp_server.h*.
NX_FTP_TIMEOUT_PERIOD	Specifies the intervals in seconds when the Server checks for Client activity. The default value is set to 60 in *nxd_ftp_server.h*.
NX_FTP_SERVER_RETRY_SECONDS	Specifies the initial timeout in seconds before retransmitting server response. The default value is 2 in *nxd_ftp_server.h*.
NX_FTP_SERVER_TRANSMIT_QUEUE_DEPTH	Specifies the maximum of depth of queued transmit packets on Server

	socket. The default value is 20 in *nxd_ftp_server.h*
NX_FTP_SERVER_RETRY_MAX	Specifies the maximum retries per packet. The default value is 10 in *nxd_ftp_server.h*.
NX_FTP_SERVER_RETRY_SHIFT	Specifies the number of bits to shift in setting the retry timeout. The default value is 2, e.g. every retry timeout is twice as long as the previous retry in *nxd_ftp_server.h*.
NX_DISABLE_ERROR_CHECKING	This option removes the basic FTP error checking. It is typically used after the application has been debugged.
NX_FTP_NO_FILEX	Defined, this option provides a stub for FileX dependencies. The FTP Client will function without any change if this option is defined. The FTP Server will need to either be modified or the user will have to create a handful of FileX services in order to function properly.
NX_FTP_CONTROL_TOS	Type of service required for the FTP TCP control requests. By default, this value is defined as NX_IP_NORMAL to indicate normal IP packet service.
NX_FTP_DATA_TOS	Type of service required for the FTP TCP data requests. By default, this value is defined as NX_IP_NORMAL to indicate normal IP packet service.
NX_FTP_FRAGMENT_OPTION	Fragment enable for FTP TCP requests. By default, this value is NX_DONT_FRAGMENT to disable FTP TCP fragmenting.

NX_FTP_CONTROL_WINDOW_SIZE	TCP Control socket window size. By default, this value is 400 bytes.
NX_FTP_DATA_WINDOW_SIZE	TCP Data socket window size. By default, this value is 2048 bytes.
NX_FTP_TIME_TO_LIVE	Specifies the number of routers this packet can pass before it is discarded. The default value is set to 0x80.
NX_FTP_USERNAME_SIZE	Specifies the number of bytes allowed in a Client supplied *username*. The default value is set to 20 *.*
NX_FTP_PASSWORD_SIZE	Specifies the number of bytes allowed in a client supplied *password*. The default value is set to 20.

Chapter 3

Description of FTP Services

This chapter contains a description of all NetX FTP services (listed below) in alphabetic order (except where IPv4 and IPv6 equivalents of the same service are paired together).

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_ftp_client_connect
Connect to FTP Server with IPv4 only

nxd_ftp_client_connect
Connect to FTP Server with IPv6 and IPv4 support

nx_ftp_client_create
Create an FTP Client instance

nx_ftp_client_delete
Delete an FTP Client instance

nx_ftp_client_set_interface
Set the FTP Client address interface

nx_ftp_client_directory_create
Create a directory on Server

nx_ftp_client_directory_default_set
Set default directory on Server

nx_ftp_client_directory_delete
Delete a directory on Server

nx_ftp_client_directory_listing_get
Get directory listing from Server

nx_ftp_client_directory_listing_continue
Continue directory listing from Server

nx_ftp_client_disconnect

Disconnect from FTP Server

nx_ftp_client_file_close
Close Client file

nx_ftp_client_file_delete
Delete file on Server

nx_ftp_client_file_open
Open Client file

nx_ftp_client_file_read
Read from file

nx_ftp_client_file_rename
Rename file on Server

nx_ftp_client_file_write
Write to file

nx_ftp_server_create
Create FTP Server with IPv4 support only

nxd_ftp_server_create
Create FTP Server with IPv4 and IPv6 support

nx_ftp_server_delete
Delete FTP Server

nx_ftp_server_start
Start FTP Server

nx_ftp_server_stop
Stop FTP Server

nx_ftp_server_set_interface
Set the FTP server address interface

nx_ftp_client_connect

Connect to an FTP Server over IPv4

Prototype

```
UINT nx_ftp_client_connect(NX_FTP_CLIENT *ftp_client_ptr,
            ULONG server_ip, CHAR *username, CHAR *password,
            ULONG wait_option);
```

Description

This service connects the previously created NetX FTP Client instance to the FTP Server at the supplied IP address.

Input Parameters

ftp_client_ptr Pointer to FTP Client control block.

server_ip IP address of FTP Server.

username Client username for authentication.

password Client password for authentication.

wait_option Defines how long the service will wait for the FTP Client connection. 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 a FTP 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 FTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful FTP connection.
NX_FTP_ERROR	(0xD0)	FTP Client connect error.

NX_FTP_NOT_DISCONNECTED	(0xD4)	FTP Client is already connected**.**
NX_PTR_ERROR	(0x16)	Invalid FTP, username, or password pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.
NX_IP_ADDRESS_ERROR	(0x21)	Invalid IP address.

Allowed From

Threads

Example

```
/* Connect the FTP Client instance "my_client" to the FTP Server at
   IP address 1.2.3.4.  */
status =  nx_ftp_client_connect(&my_client, IP_ADDRESS(1,2,3,4), NULL, NULL, 100);

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

See Also

nx_ftp_client_create, nx_ftp_client_delete, nx_ftp_client_directory_create, nx_ftp_client_disconnect, nxd_ftp_client_connect

nxd_ftp_client_connect

Connect to an FTP Server with IPv6 support

Prototype

```
UINT nxd_ftp_client_connect(NX_FTP_CLIENT *ftp_client_ptr,
            NXD_ADDRESS *server_ipduo, CHAR *username, CHAR *password,
            ULONG wait_option);
```

Description

This service connects the previously created NetX Duo FTP Client instance to the FTP Server at the supplied IP address. Both IPv4 and IPv6 networks are supported.

Input Parameters

ftp_client_ptr Pointer to FTP Client control block.

server_ipduo IP address of the FTP Server.

username Client username for authentication.

password Client password for authentication.

wait_option Defines how long the service will wait for the FTP Client connection. 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 a FTP 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 FTP Server response.

Return Values

NX_SUCCESS (0x00) Successful FTP connection.

NX_FTP_ERROR	(0xD0)	FTP Client connect error.
NX_FTP_NOT_DISCONNECTED	(0xD4)	FTP Client is already connected**.**
NX_PTR_ERROR	(0x16)	Invalid FTP, username, or password pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.
NX_IP_ADDRESS_ERROR	(0x21)	Invalid IP address.

Allowed From

Threads

Example

```
/* Connect an FTP Client instance to the FTP Server.  */

/* Set up an IPv6 address for the server here. Note this could also be an IPv4 address as
   well*/
server_ip_addr.nxd_ip_address.v6[3] = 0x106;
server_ip_addr.nxd_ip_address.v6[2] = 0x0;
server_ip_addr.nxd_ip_address.v6[1] = 0x0000f101;
server_ip_addr.nxd_ip_address.v6[0] = 0x20010db8;
server_ip_addr.nxd_ip_version = NX_IP_VERSION_V6;

status =  nxd_ftp_client_connect(&my_client, server_ip_addr, NULL, NULL, 100);


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

See Also

nx_ftp_client_create, nx_ftp_client_delete, nx_ftp_client_directory_create, nx_ftp_client_disconnect, nx_ftp_client_connect

nx_ftp_client_create

Create an FTP Client instance

Prototype

```
UINT  nx_ftp_client_create(NX_FTP_CLIENT *ftp_client_ptr,
            CHAR *ftp_client_name, NX_IP *ip_ptr, ULONG window_size,
            NX_PACKET_POOL *pool_ptr);
```

Description

This service creates an FTP Client instance.

Input Parameters

ftp_client_ptr	Pointer to FTP Client control block.
ftp_client_name	Name of FTP Client.
ip_ptr	Pointer to previously created IP instance.
window_size	Advertised window size for TCP sockets of this FTP Client.
pool_ptr	Pointer to the default packet pool for this FTP Client. Note that the minimum packet payload must be large enough to hold complete path and the file or directory name.

Return Values

NX_SUCCESS	(0x00)	Successful FTP Client create.
NX_FTP_ERROR	(0xD0)	FTP Client create error.
NX_PTR_ERROR	(0x16)	Invalid FTP, IP pointer, or packet pool pointer. password pointer.

Allowed From

Initialization and Threads

Example

```
/* Create the FTP Client instance “my_client.” */
status =  nx_ftp_client_create(&my_client, "My Client", &client_ip,
                                                    2000, &client_pool);


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

See Also

nx_ftp_client_connect, nxd_ftp_client_connect, nx_ftp_client_delete, nx_ftp_client_disconnect

nx_ftp_client_delete

Delete an FTP Client instance

Prototype

```
UINT nx_ftp_client_delete(NX_FTP_CLIENT *ftp_client_ptr);
```

Description

This service deletes an FTP Client instance.

Input Parameters

ftp_client_ptr Pointer to FTP Client control block.

Return Values

NX_SUCCESS	(0x00)	Successful FTP Client delete.
NX_FTP_ERROR	(0xD0)	FTP Client delete error.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

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

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

See Also

nx_ftp_client_connect, nxd_ftp_client_connect, nx_ftp_client_create

nx_ftp_client_directory_create

Create a directory on FTP Server

Prototype

```
UINT nx_ftp_client_directory_create(NX_FTP_CLIENT *ftp_client_ptr,
            CHAR *directory_name, ULONG wait_option);
```

Description

This service creates the specified directory on the FTP Server that is connected to the specified FTP Client.

Input Parameters

ftp_client_ptr Pointer to FTP Client control block.

directory_name Name of directory to create.

wait_option Defines how long the service will wait for the FTP directory create. 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 a FTP 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 FTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful FTP directory create.
NX_FTP_ERROR	(0xD0)	FTP directory create error.
NX_FTP_NOT_CONNECTED	(0xD3)	FTP Client is not connected.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Create the directory “my_dir” on the FTP Server connected to
   the FTP Client instance “my_client.” */
status =  nx_ftp_client_directory_create(&my_client, “my_dir”, 200);


/* If status is NX_SUCCESS the directory “my_dir” was successfully
   created.  */
```

See Also

nx_ftp_client_directory_default_set, nx_ftp_client_directory_delete, nx_ftp_client_directory_listing_get, nx_ftp_client_directory_listing_continue, nx_ftp_client_disconnect, nx_ftp_client_file_close, nx_ftp_client_file_delete, nx_ftp_client_file_open, nx_ftp_client_file_read, nx_ftp_client_file_rename, nx_ftp_client_file_write

nx_ftp_client_directory_default_set

Set default directory on FTP Server

Prototype

```
UINT nx_ftp_client_directory_default_set(NX_FTP_CLIENT *ftp_client_ptr,
                                         CHAR *directory_path, ULONG wait_option);
```

Description

This service sets the default directory on the FTP Server that is connected to the specified FTP Client. This default directory applies only to this client's connection.

Input Parameters

ftp_client_ptr Pointer to FTP Client control block.

directory_path Name of directory path to set.

wait_option Defines how long the service will wait for the FTP default directory set. 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 a FTP 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 FTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful FTP default set.
NX_FTP_ERROR	(0xD0)	FTP default set error.
NX_FTP_NOT_CONNECTED	(0xD3)	FTP Client is not connected.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Set the default directory to “my_dir” on the FTP Server connected to
   the FTP Client instance “my_client.” */
status =  nx_ftp_client_directory_default_set(&my_client, “my_dir”, 200);

/* If status is NX_SUCCESS the directory “my_dir” is the default directory.  */
```

See Also

nx_ftp_client_directory_create, nx_ftp_client_directory_delete, nx_ftp_client_directory_listing_get, nx_ftp_client_directory_listing_continue, nx_ftp_client_disconnect, nx_ftp_client_file_close, nx_ftp_client_file_delete, nx_ftp_client_file_open, nx_ftp_client_file_read, nx_ftp_client_file_rename, nx_ftp_client_file_write

nx_ftp_client_directory_delete

Delete directory on FTP Server

Prototype

```
UINT nx_ftp_client_directory_delete(NX_FTP_CLIENT *ftp_client_ptr,
            CHAR *directory_name, ULONG wait_option);
```

Description

This service deletes the specified directory on the FTP Server that is connected to the specified FTP Client.

Input Parameters

ftp_client_ptr Pointer to FTP Client control block.

directory_name Name of directory to delete.

wait_option Defines how long the service will wait for the FTP directory delete. 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 a FTP 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 FTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful FTP directory delete.
NX_FTP_ERROR	(0xD0)	FTP directory delete error.
NX_FTP_NOT_CONNECTED	(0xD3)	FTP Client is not connected.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Delete directory “my_dir” on the FTP Server connected to
   the FTP Client instance “my_client.” */
status =  nx_ftp_client_directory_delete(&my_client, “my_dir”, 200);

/* If status is NX_SUCCESS the directory “my_dir” is deleted.  */
```

See Also

nx_ftp_client_directory_create, nx_ftp_client_directory_default_set, nx_ftp_client_directory_listing_get, nx_ftp_client_directory_listing_continue, nx_ftp_client_disconnect, nx_ftp_client_file_close, nx_ftp_client_file_delete, nx_ftp_client_file_open, nx_ftp_client_file_read, nx_ftp_client_file_rename, nx_ftp_client_file_write

nx_ftp_client_directory_listing_get

Get directory listing from FTP Server

Prototype

```
UINT nx_ftp_client_directory_listing_get(NX_FTP_CLIENT *ftp_client_ptr,
            CHAR *directory_name, NX_PACKET **packet_ptr,
            ULONG wait_option);
```

Description

This service gets the contents of the specified directory on the FTP Server that is connected to the specified FTP Client. The supplied packet pointer will contain one or more directory entries. Each entry is separated by a <cr/lf> combination. The ***nx_ftp_client_directory_listing_continue*** should be called to complete the directory get operation.

Input Parameters

ftp_client_ptr Pointer to FTP Client control block.

directory_name Name of directory to get contents of.

packet_ptr Pointer to destination packet pointer. If successful, the packet payload will contain one or more directory entries.

wait_option Defines how long the service will wait for the FTP directory listing. 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 a FTP 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 FTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful FTP directory listing.
NX_FTP_ERROR	(0xD0)	FTP directory listing error.
NX_FTP_NOT_CONNECTED	(0xD3)	FTP Client is not connected.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Get the contents of directory “my_dir” on the FTP Server connected to
   the FTP Client instance “my_client.” */
status =  nx_ftp_client_directory_listing_get(&my_client, “my_dir”, &my_packet,
                                                200);


/* If status is NX_SUCCESS, one or more entries of the directory “my_dir”
   can be found in “my_packet”.  */
```

See Also

nx_ftp_client_directory_create, nx_ftp_client_directory_default_set, nx_ftp_client_directory_delete, nx_ftp_client_directory_listing_continue, nx_ftp_client_disconnect, nx_ftp_client_file_close, nx_ftp_client_file_delete, nx_ftp_client_file_open, nx_ftp_client_file_read, nx_ftp_client_file_rename, nx_ftp_client_file_write

nx_ftp_client_directory_listing_continue

Continue directory listing from FTP Server

Prototype

```
UINT nx_ftp_client_directory_listing_continue(NX_FTP_CLIENT
            *ftp_client_ptr, NX_PACKET **packet_ptr,
            ULONG wait_option);
```

Description

This service continues getting the contents of the specified directory on the FTP Server that is connected to the specified FTP Client. It should have been immediately preceded by a call to ***nx_ftp_client_directory_listing_get***. If successful, the supplied packet pointer will contain one or more directory entries. This routine should be called until an NX_FTP_END_OF_LISTING status is received.

Input Parameters

ftp_client_ptr Pointer to FTP Client control block.

packet_ptr Pointer to destination packet pointer. If successful, the packet payload will contain one or more directory entries, separated by a <cr/lf>.

wait_option Defines how long the service will wait for the FTP directory listing. 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 a FTP 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 FTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful FTP directory listing.
NX_FTP_ERROR	(0xD0)	FTP directory listing error.
NX_FTP_END_OF_LISTING	(0xD8)	No more entries in this directory.
NX_FTP_NOT_CONNECTED	(0xD3)	FTP Client is not connected.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Continue getting the contents of directory "my_dir" on the FTP Server
   connected to the FTP Client instance "my_client." */
status =  nx_ftp_client_directory_listing_continue(&my_client, &my_packet,
                                                   200);


/* If status is NX_SUCCESS, one or more entries of the directory "my_dir"
   can be found in "my_packet".  */
```

See Also

nx_ftp_client_directory_create, nx_ftp_client_directory_default_set, nx_ftp_client_directory_delete, nx_ftp_client_directory_listing_get, nx_ftp_client_disconnect, nx_ftp_client_file_close, nx_ftp_client_file_delete, nx_ftp_client_file_open, nx_ftp_client_file_read, nx_ftp_client_file_rename, nx_ftp_client_file_write

nx_ftp_client_disconnect

Disconnect from FTP Server

Prototype

```
UINT nx_ftp_client_disconnect(NX_FTP_CLIENT *ftp_client_ptr,
                                                ULONG wait_option);
```

Description

This service disconnects a previously established FTP Server connection with the specified FTP Client.

Input Parameters

ftp_client_ptr Pointer to FTP Client control block.

wait_option Defines how long the service will wait for the FTP 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 a FTP 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 FTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful FTP disconnect.
NX_FTP_ERROR	(0xD0)	FTP Client disconnect error.
NX_FTP_NOT_CONNECTED	(0xD3)	FTP Client is not connected.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Disconnect “my_client” from the FTP Server. */
status =  nx_ftp_client_disconnect(&my_client, 200);

/* If status is NX_SUCCESS, “my_client” has been disconnected.  */
```

See Also

nx_ftp_client_directory_create, nx_ftp_client_directory_default_set, nx_ftp_client_directory_delete, nx_ftp_client_directory_listing_get, nx_ftp_client_directory_listing_continue, nx_ftp_client_file_close, nx_ftp_client_file_delete, nx_ftp_client_file_open, nx_ftp_client_file_read, nx_ftp_client_file_rename, nx_ftp_client_file_write

nx_ftp_client_file_close

Close Client file

Prototype

```
UINT nx_ftp_client_file_close(NX_FTP_CLIENT *ftp_client_ptr,
                                                ULONG wait_option);
```

Description

This service closes a previously opened file on the FTP Server.

Input Parameters

ftp_client_ptr Pointer to FTP Client control block.

wait_option Defines how long the service will wait for the FTP Client file close. 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 a FTP 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 FTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful FTP file close.
NX_FTP_ERROR	(0xD0)	FTP Client file close error.
NX_FTP_NOT_CONNECTED	(0xD3)	FTP Client is not connected.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Close previously opened file of client “my_client” on the FTP Server. */
status =  nx_ftp_client_file_close(&my_client, 200);

/* If status is NX_SUCCESS, the file opened previously in the “my_client” FTP
   connection has been closed.  */
```

See Also

nx_ftp_client_directory_create, nx_ftp_client_directory_default_set, nx_ftp_client_directory_delete, nx_ftp_client_directory_listing_get, nx_ftp_client_directory_listing_continue, nx_ftp_client_disconnect, nx_ftp_client_file_delete, nx_ftp_client_file_open, nx_ftp_client_file_read, nx_ftp_client_file_rename, nx_ftp_client_file_write

nx_ftp_client_file_delete

Delete file on FTP Server

Prototype

```
UINT nx_ftp_client_file_delete(NX_FTP_CLIENT *ftp_client_ptr,
                  CHAR *file_name, ULONG wait_option);
```

Description

This service deletes the specified file on the FTP Server.

Input Parameters

ftp_client_ptr Pointer to FTP Client control block.

file_name Name of file to delete.

wait_option Defines how long the service will wait for the FTP Client file delete. 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 a FTP 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 FTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful FTP file delete.
NX_FTP_ERROR	(0xD0)	FTP Client file delete error.
NX_FTP_NOT_CONNECTED	(0xD3)	FTP Client is not connected.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Delete the file “my_file.txt” on the FTP Server using the previously
   connected client “my_client.” */
status =  nx_ftp_client_file_delete(&my_client, “my_file.txt”, 200);


/* If status is NX_SUCCESS, the file “my_file.txt” on the FTP Server is
   deleted.  */
```

See Also

nx_ftp_client_directory_create, nx_ftp_client_directory_default_set, nx_ftp_client_directory_delete, nx_ftp_client_directory_listing_get, nx_ftp_client_directory_listing_continue, nx_ftp_client_disconnect, nx_ftp_client_file_close, nx_ftp_client_file_open, nx_ftp_client_file_read, nx_ftp_client_file_rename, nx_ftp_client_file_write

nx_ftp_client_file_open

Opens file on FTP Server

Prototype

```
UINT nx_ftp_client_file_open(NX_FTP_CLIENT *ftp_client_ptr,
            CHAR *file_name, UINT open_type, ULONG wait_option);
```

Description

This service opens the specified file – for reading or writing – on the FTP Server previously connected to the specified Client instance.

Input Parameters

ftp_client_ptr Pointer to FTP Client control block.

file_name Name of file to open.

open_type Either **NX_FTP_OPEN_FOR_READ** or **NX_FTP_OPEN_FOR_WRITE**.

wait_option Defines how long the service will wait for the FTP Client file open. 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 a FTP 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 FTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful FTP file open.
NX_FTP_ERROR	(0xD0)	FTP Client file open error.
NX_OPTION_ERROR	(0x0A)	Invalid open type.

NX_FTP_NOT_CONNECTED (0xD3)		FTP Client is not connected.
NX_FTP_NOT_CLOSED	(0xD6)	FTP Client is already opened.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Open the file “my_file.txt” for reading on the FTP Server using the previously
   connected client “my_client.” */
status =  nx_ftp_client_file_open(&my_client, “my_file.txt”, NX_FTP_OPEN_FOR_READ,
                                  200);


/* If status is NX_SUCCESS, the file “my_file.txt” on the FTP Server is
   open for reading.  */
```

See Also

nx_ftp_client_directory_create, nx_ftp_client_directory_default_set, nx_ftp_client_directory_delete, nx_ftp_client_directory_listing_get, nx_ftp_client_directory_listing_continue, nx_ftp_client_disconnect, nx_ftp_client_file_close, nx_ftp_client_file_delete, nx_ftp_client_file_read, nx_ftp_client_file_rename, nx_ftp_client_file_write

nx_ftp_client_file_read

Read from file

Prototype

```
UINT nx_ftp_client_file_read(NX_FTP_CLIENT *ftp_client_ptr,
                        NX_PACKET **packet_ptr, ULONG wait_option);
```

Description

This service reads a packet from a previously opened file. It should be called repetitively until a status of NX_FTP_END_OF_FILE is received.

Input Parameters

ftp_client_ptr Pointer to FTP Client control block.

packet_ptr Pointer to destination for the data packet pointer to be stored. If successful, the packet some or all the contains of the file.

wait_option Defines how long the service will wait for the FTP Client file read. 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 a FTP 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 FTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful FTP file read.
NX_FTP_ERROR	(0xD0)	FTP Client file read error.
NX_FTP_NOT_OPEN	(0xD5)	FTP Client is not opened.
NX_FTP_END_OF_FILE	(0xD7)	End of file condition.

NX_PTR_ERROR	(0x16)	Invalid FTP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Read a packet of data from file “my_file.txt” that was previously opened
   from the client “my_client.” */
status =  nx_ftp_client_file_read(&my_client, &my_packet, 200);

/* If status is NX_SUCCESS, the packet “my_packet” contains the next bytes
   from the file.  */
```

See Also

nx_ftp_client_directory_create, nx_ftp_client_directory_default_set, nx_ftp_client_directory_delete, nx_ftp_client_directory_listing_get, nx_ftp_client_directory_listing_continue, nx_ftp_client_disconnect, nx_ftp_client_file_close, nx_ftp_client_file_delete, nx_ftp_client_file_open, nx_ftp_client_file_rename, nx_ftp_client_file_write

nx_ftp_client_file_rename

Rename file on FTP Server

Prototype

```
UINT nx_ftp_client_file_rename(NX_FTP_CLIENT *ftp_ptr, CHAR *filename,
                        CHAR *new_filename, ULONG wait_option);
```

Description

This service renames a file on the FTP Server.

Input Parameters

ftp_client_ptr Pointer to FTP Client control block.

filename Current name of file.

new_filename New name for file.

wait_option Defines how long the service will wait for the FTP Client file rename. 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 a FTP 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 FTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful FTP file rename.
NX_FTP_ERROR	(0xD0)	FTP Client file rename error.
NX_FTP_NOT_CONNECTED	(0xD3)	FTP Client is not connected.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Rename file “my_file.txt” to “new_file.txt” on the previously connected
   Client instance “my_client.” */
status =  nx_ftp_client_file_rename(&my_client, “my_file.txt”, “new_file.txt”,
                    200);

/* If status is NX_SUCCESS, the file has been renamed.  */
```

See Also

nx_ftp_client_directory_create, nx_ftp_client_directory_default_set, nx_ftp_client_directory_delete, nx_ftp_client_directory_listing_get, nx_ftp_client_directory_listing_continue, nx_ftp_client_disconnect, nx_ftp_client_file_close, nx_ftp_client_file_delete, nx_ftp_client_file_open, nx_ftp_client_file_read, nx_ftp_client_file_write

nx_ftp_client_file_write

Write to file

Prototype

```
UINT nx_ftp_client_file_write(NX_FTP_CLIENT *ftp_client_ptr,
                        NX_PACKET *packet_ptr, ULONG wait_option);
```

Description

This service writes a packet of data to the previously opened file on the FTP Server.

Input Parameters

ftp_client_ptr Pointer to FTP Client control block.

packet_ptr Packet pointer containing data to write.

wait_option Defines how long the service will wait for the FTP Client file write. 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 a FTP 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 FTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful FTP file write.
NX_FTP_ERROR	(0xD0)	FTP Client file write error.
NX_FTP_NOT_OPEN	(0xD5)	FTP Client is not opened.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Write the data contained in “my_packet” to the previously opened file
   “my_file.txt”.  */
status =  nx_ftp_client_file_write(&my_client, my_packet, 200);

/* If status is NX_SUCCESS, the file has been written to.  */
```

See Also

nx_ftp_client_directory_create, nx_ftp_client_directory_default_set, nx_ftp_client_directory_delete, nx_ftp_client_directory_listing_get, nx_ftp_client_directory_listing_continue, nx_ftp_client_disconnect, nx_ftp_client_file_close, nx_ftp_client_file_delete, nx_ftp_client_file_open, nx_ftp_client_file_read, nx_ftp_client_file_rename

nx_ftp_server_create

Create FTP Server

Prototype

```
UINT nx_ftp_server_create(NX_FTP_SERVER *ftp_server_ptr,
            CHAR *ftp_server_name, NX_IP *ip_ptr,
            FX_MEDIA *media_ptr, VOID *stack_ptr, ULONG stack_size,
            NX_PACKET_POOL *pool_ptr,
            UINT (*ftp_login)(struct NX_FTP_SERVER_STRUCT
                   *ftp_server_ptr, ULONG client_ip_address,
                  UINT client_port, CHAR *name, CHAR *password,
                  CHAR *extra_info),
            UINT (*ftp_logout)(struct NX_FTP_SERVER_STRUCT
                  *ftp_server_ptr,   ULONG client_ip_address,
                  UINT client_port, CHAR *name, CHAR *password,
                  CHAR *extra_info));
```

Description

This service creates an FTP Server instance on the specified and previously created NetX IP instance. Note the FTP Server needs to be started with a call to ***nx_ftp_server_start*** for it to begin operation.

Input Parameters

ftp_server_ptr	Pointer to FTP Server control block.
ftp_server_name	Name of FTP Server.
ip_ptr	Pointer to associated NetX IP instance. Note there can only be one FTP Server for an IP instance.
media_ptr	Pointer to associated FileX media instance.
stack_ptr	Pointer to memory for the internal FTP Server thread's stack area.
stack_size	Size of stack area specified by *stack_ptr*.
pool_ptr	Pointer to default NetX packet pool. Note the payload size of packets in the pool must be large enough to accommodate the largest filename/path.
ftp_login	Function pointer to application's login function. This function is supplied the username and password from the Client requesting a connection, and the Client

address in the ULONG data type. If this is valid, the application's login function should return NX_SUCCESS.

ftp_logout Function pointer to application's logout function. This function is supplied the username and password from the Client requesting a disconnection, and the Client address in the ULONG data type. If this is valid, the application's login function should return NX_SUCCESS.

Return Values

NX_SUCCESS	(0x00)	Successful FTP Server create.
NX_FTP_ERROR	(0xD0)	FTP Server create error.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.

Allowed From

Initialization and Threads

Example

```
/* Create the FTP Server "my_server" on the IP instance "ip_0" using the
   "ram_disk" media.  */
status =  nx_ftp_server_create(&my_server, "My Server Name", &ip_0, &ram_disk,
                        stack_ptr, stack_size, &pool_0,
                        my_login, my_logout);

/* If status is NX_SUCCESS, the FTP Server has been created.  */
```

See Also

nx_ftp_server_delete, nx_ftp_server_start, nx_ftp_server_stop

nxd_ftp_server_create

Create FTP Server with IPv4 and IPv6 support

Prototype

```
UINT nxd_ftp_server_create(NX_FTP_SERVER *ftp_server_ptr,
            CHAR *ftp_server_name, NX_IP *ip_ptr,
            FX_MEDIA *media_ptr, VOID *stack_ptr, ULONG stack_size,
            NX_PACKET_POOL *pool_ptr,
            UINT (*ftp_login_duo)(struct NX_FTP_SERVER_STRUCT
                   *ftp_server_ptr, NXD_ADDRESS *client_ip_address,
                  UINT client_port, CHAR *name, CHAR *password,
                  CHAR *extra_info),
            UINT (*ftp_logout_duo)(struct NX_FTP_SERVER_STRUCT
                  *ftp_server_ptr,   NXD_ADDRESS *client_ip_address,
                  UINT client_port, CHAR *name, CHAR *password,
                  CHAR *extra_info));
```

Description

This service creates an FTP Server instance on the specified and previously created NetX IP instance. Note the FTP Server still needs to be started with a call to ***nx_ftp_server_start*** for it to begin operation after the Server is created.

Input Parameters

ftp_server_ptr	Pointer to FTP Server control block.
ftp_server_name	Name of FTP Server.
ip_ptr	Pointer to associated NetX IP instance. Note there can only be one FTP Server for an IP instance.
media_ptr	Pointer to associated FileX media instance.
stack_ptr	Pointer to memory for the internal FTP Server thread's stack area.
stack_size	Size of stack area specified by *stack_ptr*.
pool_ptr	Pointer to default NetX packet pool. Note the payload size of packets in the pool must be large enough to accommodate the largest filename/path.
ftp_login_duo	Function pointer to application's login function. This function is supplied the username and password from the Client requesting a connection, and a pointer to the Client address in the NXD_ADDRESS data type. If this is

valid, the application's login function should return NX_SUCCESS.

ftp_logout_duo Function pointer to application's logout function. This function is supplied the username and password from the Client requesting a disconnection, and a pointer to the Client address in the NXD_ADDRESS data type. If this is valid, the application's login function should return NX_SUCCESS.

Return Values

NX_SUCCESS	(0x00)	Successful FTP Server create.
NX_FTP_ERROR	(0xD0)	FTP Server create error.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.

Allowed From

Initialization and Threads

Example

```
/* Create the FTP Server "my_server" on the IP instance "ip_0" using the
   "ram_disk" media.  */
status =  nxd_ftp_server_create(&my_server, "My Server Name", &ip_0, &ram_disk,
                        stack_ptr, stack_size, &pool_0,
                        my_duo_login, my_duo_logout);

/* If status is NX_SUCCESS, the FTP Server has been created.  */
```

See Also

nx_ftp_server_create, nx_ftp_server_delete, nx_ftp_server_start, nx_ftp_server_stop

nx_ftp_server_delete

Delete FTP Server

Prototype

```
UINT nx_ftp_server_delete(NX_FTP_SERVER *ftp_server_ptr);
```

Description

This service deletes a previously created FTP Server instance.

Input Parameters

ftp_server_ptr Pointer to FTP Server control block.

Return Values

NX_SUCCESS	(0x00)	Successful FTP Server delete.
NX_FTP_ERROR	(0xD0)	FTP Server delete error.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Delete the FTP Server "my_server". */
status =  nx_ftp_server_delete(&my_server);

/* If status is NX_SUCCESS, the FTP Server has been deleted.  */
```

See Also

nx_ftp_server_create, nx_ftp_server_start, nx_ftp_server_stop

nx_ftp_server_start

Start FTP Server

Prototype

```
UINT nx_ftp_server_start(NX_FTP_SERVER *ftp_server_ptr);
```

Description

This service starts a previously created FTP Server instance.

Input Parameters

ftp_server_ptr Pointer to FTP Server control block.

Return Values

NX_SUCCESS	(0x00)	Successful FTP Server start.
NX_FTP_ERROR	(0xD0)	FTP Server start error.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.

Allowed From

Initialization and Threads

Example

```
/* Start the FTP Server "my_server". */
status =  nx_ftp_server_start(&my_server);

/* If status is NX_SUCCESS, the FTP Server has been started.  */
```

See Also

nx_ftp_server_create, nx_ftp_server_delete, nx_ftp_server_stop

nx_ftp_server_stop

Stop FTP Server

Prototype

```
UINT nx_ftp_server_stop(NX_FTP_SERVER *ftp_server_ptr);
```

Description

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

Input Parameters

ftp_server_ptr	Pointer to FTP Server control block.

Return Values

NX_SUCCESS	(0x00)	Successful FTP Server stop.
NX_FTP_ERROR	(0xD0)	FTP Server stop error.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Stop the FTP Server "my_server". */
status =  nx_ftp_server_stop(&my_server);

/* If status is NX_SUCCESS, the FTP Server has been stopped.  */
```

See Also

nx_ftp_server_create, nx_ftp_server_delete, nx_ftp_server_start

nx_ftp_server_set_interface

Set the FTP Server interface

Prototype

```
UINT nx_ftp_server_set_interface(NX_IP *ip_ptr,
                        NX_FTP_SERVER*ftp_server_ptr,
                        NXD_ADDRESS *server_ip_address);
```

Description

This service sets the FTP Server interface based on the input Server global address. The default setting for the Server FTP interface is the primary interface at index 0 in the IP interface table, and primary global address at index 1 in the IP task address table as the interface parameters.

Input Parameters

ip_ptr	Pointer to IP control block.
ftp_server_ptr	Pointer to FTP Server control block.
server_ip_address	Pointer to FTP Server global address

Return Values

NX_SUCCESS	(0x00)	Successful FTP interface set.
NX_PTR_ERROR	(0x16)	Invalid input pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.
NX_NOT_ENABLED	(0x11)	Available only if IPv6 enabled

Allowed From

Application code

Example

```
/* Set the FTP Client interface using the primary global address." */

NXD_ADDRESS server _ip_address;

server_ip_address.nxd_ip_address.v6[3] = 0x101;
server _ip_address.nxd_ip_address.v6[2] = 0x0;
server _ip_address.nxd_ip_address.v6[1] = 0x0000f101;
server _ip_address.nxd_ip_address.v6[0] = 0x20010db8;
server _ip_address.nxd_ip_version = NX_IP_VERSION_V6;

status =  nx_ftp_ server _set_interface(ip_ptr, &ftp_ server, & server _ip_address);
```

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

See Also

nx_ftp_server_connect, nxd_ftp_server_connect, nx_ftp_server_create