



File Transfer Protocol (FTP)

User Guide

Express Logic, Inc.

858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259

www.expresslogic.com

©2002-2013 by Express Logic, Inc.

All rights reserved. This document and the associated NetX software are the sole property of Express Logic, Inc. Each contains proprietary information of Express Logic, Inc. Reproduction or duplication by any means of any portion of this document without the prior written consent of Express Logic, Inc. is expressly forbidden. Express Logic, Inc. reserves the right to make changes to the specifications described herein at any time and without notice in order to improve design or reliability of NetX. The information in this document has been carefully checked for accuracy; however, Express Logic, Inc. makes no warranty pertaining to the correctness of this document.

Trademarks

NetX, Piconet, and UDP Fast Path are trademarks of Express Logic, Inc. ThreadX is a registered trademark of Express Logic, Inc.

All other product and company names are trademarks or registered trademarks of their respective holders.

Warranty Limitations

Express Logic, Inc. makes no warranty of any kind that the NetX products will meet the USER's requirements, or will operate in the manner specified by the USER, or that the operation of the NetX products will operate uninterrupted or error free, or that any defects that may exist in the NetX products will be corrected after the warranty period. Express Logic, Inc. makes no warranties of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose, with respect to the NetX products. No oral or written information or advice given by Express Logic, Inc., its dealers, distributors, agents, or employees shall create any other warranty or in any way increase the scope of this warranty, and licensee may not rely on any such information or advice.

Part Number: 000-1052

Revision 5.3

Contents

Chapter 1 Introduction to FTP	4
FTP Requirements	4
FTP Constraints	4
FTP File Names	5
FTP Client Commands	5
FTP Server Responses	5
FTP Communication	6
FTP Authentication	8
FTP Multi-Thread Support	8
FTP RFCs	8
Chapter 2 Installation and Use of FTP	9
Product Distribution	9
FTP Installation	9
Using FTP	9
Small Example System	10
Configuration Options	15
Chapter 3 Description of FTP Services	18
nx_ftp_client_connect	20
nx_ftp_client_create	22
nx_ftp_client_delete	24
nx_ftp_client_directory_create	25
nx_ftp_client_directory_default_set	27
nx_ftp_client_directory_delete	29
nx_ftp_client_directory_listing_get	31
nx_ftp_client_directory_listing_continue	33
nx_ftp_client_disconnect	35
nx_ftp_client_file_close	37
nx_ftp_client_file_delete	39
nx_ftp_client_file_open	41
nx_ftp_client_file_read	43
nx_ftp_client_file_rename	45
nx_ftp_client_file_write	47
nx_ftp_server_create	49
nx_ftp_server_delete	51
nx_ftp_server_start	52
nx_ftp_server_stop	53

Chapter 1

Introduction to 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.

FTP Requirements

In order to function properly, the NetX FTP package requires that a NetX IP instance has already been created. In addition, TCP must be enabled on that same IP instance. 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. The FTP Server is also designed for use with the FileX embedded file system. If FileX is not available, the user may port the portions of FileX used to their own environment. This is discussed in later sections of this guide.

FTP Constraints

The FTP standard has many options regarding the representation of file data. Similar to Unix implementations, NetX FTP assumes the following file format constraints:

File Type:	Binary
File Format:	Nonprint Only
File Structure:	File Structure Only
Transmission Mode:	Stream Mode 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:

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

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

FTP Server Responses

The FTP Server utilizes the *well-known TCP port 21* to field Client command requests. Once the FTP Server processes the Client command, it returns a 3-digit numeric response in ASCII followed by an optional ASCII string. The numeric response is used by the FTP Client software to determine whether the operation succeeded or failed. The following lists various FTP Server responses to Client commands:

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

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:

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 "PORT" 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. 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 "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 ***nx_ftp_server_create*** function and called from the password processing. 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 FTP is compliant with RFC959 and related RFCs.

Chapter 2

Installation and Use of FTP

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

Product Distribution

FTP for NetX 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:

<code>nx_ftp.h</code>	Header file for FTP for NetX
<code>nx_ftp_client.c</code>	C Source file for FTP Client for NetX
<code>nx_ftp_server.c</code>	C Source file for FTP Server for NetX
<code>filex_stub.h</code>	Stub file if FileX is not present
<code>nx_ftp.pdf</code>	PDF description of FTP for NetX
<code>demo_netx_ftp.c</code>	FTP demonstration system

FTP Installation

In order to use FTP for NetX, the entire distribution mentioned previously should be copied to the same directory where NetX is installed. For example, if NetX is installed in the directory “`\threadx\arm7\green`” then the `nx_ftp.h`, `nx_ftp_client.c`, and `nx_ftp_server.c` files should be copied into this directory.

Using FTP

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

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

Small Example System

An example of how easy it is to use NetX FTP is described in Figure 1.1 that appears below.

Note this is for a host device with a single network interface.

In this example, the FTP include file *nx_ftp.h* is brought in at line 8. Next, the FTP Server is created in “*tx_application_define*” at line 136. Note that the FTP Server control block “*Server*” was defined as a global variable at line 26 previously. After successful creation, an FTP Server is started at line 146. At line 183 the FTP Client is created. And finally, the Client writes the file at line 223 and reads the file back at line 245.

```

0001 /* This is a small demo of FTP on the high-performance NetX TCP/IP stack.
0002      This demo relies on ThreadX, NetX, and FileX to show a simple file
0003      transfer from the client and then back to the server. */
0004
0005 #include      "tx_api.h"
0006 #include      "nx_api.h"
0007 #include      "fx_api.h"
0008 #include      "nx_ftp.h"
0009
0010 #define      DEMO_STACK_SIZE          2048
0011
0012
0013 /* Define the ThreadX, NetX, and FileX object control blocks... */
0014
0015 TX_THREAD      client_thread;
0016 NX_PACKET_POOL server_pool;
0017 NX_IP           server_ip;
0018 NX_PACKET_POOL client_pool;
0019 NX_IP           client_ip;
0020 FX_MEDIA        ram_disk;
0021
0022
0023 /* Define the NetX FTP object control blocks. */
0024
0025 NX_FTP_CLIENT   client;
0026 NX_FTP_SERVER   server;
0027
0028
0029 /* Define the counters used in the demo application... */
0030
0031 ULONG           error_counter;
0032
0033
0034 /* Define the memory area for the FileX RAM disk. */
0035
0036 UCHAR           ram_disk_memory[32000];
0037
0038
0039 /* Define function prototypes. */
0040
0041 VOID            _fx_ram_driver(FX_MEDIA *media_ptr);
0042 VOID            _nx_ram_network_driver(NX_IP_DRIVER *driver_req_ptr);
0043 void            client_thread_entry(ULONG thread_input);
0044
0045 /* Define server login/logout functions. These will validate the client
0046 login request. */
0047
0048 UINT            server_login(struct NX_FTP_SERVER_STRUCT *ftp_server_ptr,
0049                             ULONG client_ip_address, UINT client_port, CHAR *name,
0050                             CHAR *password, CHAR *extra_info);
0051 UINT            server_logout(struct NX_FTP_SERVER_STRUCT *ftp_server_ptr,
0052                              ULONG client_ip_address, UINT client_port, CHAR *name,
0053                              CHAR *password, CHAR *extra_info);
0054
0055
0056 /* Define main entry point. */

```

```

0057
0058 int main()
0059 {
0060
0061     /* Enter the ThreadX kernel. */
0062     tx_kernel_enter();
0063 }
0064
0065
0066 /* Define what the initial system looks like. */
0067
0068 void    tx_application_define(void *first_unused_memory)
0069 {
0070
0071     UINT    status;
0072     UCHAR    *pointer;
0073
0074
0075     /* Setup the working pointer. */
0076     pointer = (UCHAR *) first_unused_memory;
0077
0078     /* Create the main FTP demo thread. */
0079     status = tx_thread_create(&client_thread, "thread 0",
0080                             client_thread_entry, 0, pointer, DEMO_STACK_SIZE,
0081                             17, 17, TX_NO_TIME_SLICE, TX_AUTO_START);
0082     pointer += DEMO_STACK_SIZE ;
0083
0084     /* Check for errors. */
0085     if (status)
0086         error_counter++;
0087
0088     /* Open the RAM disk. */
0089     status = fx_media_open(&ram_disk, "RAM DISK", _fx_ram_driver,
0090                           ram_disk_memory, pointer, 4096);
0091     pointer += 4096 ;
0092
0093     /* Check for errors. */
0094     if (status)
0095         error_counter++;
0096
0097     /* Initialize NetX. */
0098     nx_system_initialize();
0099
0100     /* Create the packet pool for the FTP Server. */
0101     status = nx_packet_pool_create(&server_pool, "NetX Server Packet Pool",
0102                                   256, pointer, 8192);
0103     pointer = pointer + 8192;
0104
0105     /* Check for errors. */
0106     if (status)
0107         error_counter++;
0108
0109     /* Create the IP instance for the FTP Server. */
0110     status = nx_ip_create(&server_ip, "NetX Server IP Instance",
0111                           IP_ADDRESS(1,2,3,4), 0xFFFFFFFF0UL,
0112                           &server_pool, _nx_ram_network_driver,
0113                           pointer, 2048, 1);
0114     pointer = pointer + 2048;
0115
0116     /* Check for errors. */
0117     if (status)
0118         error_counter++;
0119
0120     /* Enable ARP and supply ARP cache memory for IP Instance 0. */
0121     status = nx_arp_enable(&server_ip, (void *) pointer, 1024);
0122     pointer = pointer + 1024;
0123
0124     /* Check for errors. */
0125     if (status)
0126         error_counter++;
0127
0128     /* Enable TCP. */
0129     status = nx_tcp_enable(&server_ip);
0130
0131     /* Check for errors. */
0132     if (status)
0133         error_counter++;
0134
0135     /* Create the FTP server. */
0136     status = nx_ftp_server_create(&server, "FTP Server Instance",
0137                                   &server_ip, &ram_disk, pointer, DEMO_STACK_SIZE, &server_pool,

```

```

0138         server_login, server_logout);
0139     pointer = pointer + DEMO_STACK_SIZE;
0140
0141     /* Check for errors. */
0142     if (status)
0143         error_counter++;
0144
0145     /* Start the FTP server. */
0146     status = nx_ftp_server_start(&server);
0147
0148     /* Check for errors. */
0149     if (status)
0150         error_counter++;
0151
0152     /* Create a packet pool for the FTP client. */
0153     status = nx_packet_pool_create(&client_pool, "NetX Client Packet Pool",
0154                                   256, pointer, 8192);
0155     pointer = pointer + 8192;
0156
0157     /* Create an IP instance for the FTP client. */
0158     status = nx_ip_create(&client_ip, "NetX Client IP Instance",
0159                          IP_ADDRESS(1, 2, 3, 5), 0xFFFFFFFF0UL,
0160                          &client_pool, _nx_ram_network_driver,
0161                          pointer, 2048, 1);
0162     pointer = pointer + 2048;
0163
0164     /* Enable ARP and supply ARP cache memory for IP Instance 1. */
0165     status = nx_arp_enable(&client_ip, (void *) pointer, 1024);
0166     pointer = pointer + 1024;
0167
0168     /* Enable TCP for client IP instance. */
0169     status = nx_tcp_enable(&client_ip);
0170
0171     return;
0172 }
0173
0174 /* Define the FTP client thread. */
0175 void client_thread_entry(ULONG thread_input)
0176 {
0177     NX_PACKET *my_packet;
0178     UINT status;
0179
0180
0181     /* Create an FTP client. */
0182     status = nx_ftp_client_create(&client, "New Client", &client_ip, 2000,
0183                                   &client_pool);
0184
0185     /* Check status. */
0186     if (status)
0187         error_counter++;
0188
0189     /* Connect with FTP server. */
0190     status = nx_ftp_client_connect(&client, IP_ADDRESS(1,2,3,4), NULL, NULL,
0191                                   100);
0192
0193     /* Check status. */
0194     if (status)
0195         error_counter++;
0196
0197     /* Open an FTP file for writing. */
0198     status = nx_ftp_client_file_open(&client, "test.txt",
0199                                       NX_FTP_OPEN_FOR_WRITE, 100);
0200
0201     /* Check status. */
0202     if (status)
0203         error_counter++;
0204
0205     /* Allocate an FTP packet. */
0206     status = nx_packet_allocate(&client_pool, &my_packet, NX_TCP_PACKET,
0207                                100);
0208
0209     /* Check status. */
0210     if (status)
0211         error_counter++;
0212
0213     /* Write ABCs into the packet payload! */
0214     memcpy(my_packet -> nx_packet_prepend_ptr,
0215            "ABCDEFGHJKLMNOPQRSTUVWXYZ ", 28);
0216
0217     /* Adjust the write pointer. */
0218

```

```

0219 my_packet -> nx_packet_length = 28;
0220 my_packet -> nx_packet_append_ptr = my_packet->nx_packet_prepend_ptr+28;
0221
0222 /* Write the packet to the file test.txt. */
0223 status = nx_ftp_client_file_write(&client, my_packet, 100);
0224
0225 /* Check status. */
0226 if (status)
0227     error_counter++;
0228
0229 /* Close the file. */
0230 status = nx_ftp_client_file_close(&client, 100);
0231
0232 /* Check status. */
0233 if (status)
0234     error_counter++;
0235
0236 /* Now open the same file for reading. */
0237 status = nx_ftp_client_file_open(&client, "test.txt",
0238                                 NX_FTP_OPEN_FOR_READ, 100);
0239
0240 /* Check status. */
0241 if (status)
0242     error_counter++;
0243
0244 /* Read the file. */
0245 status = nx_ftp_client_file_read(&client, &my_packet, 100);
0246
0247 /* Check status. */
0248 if (status != NX_SUCCESS)
0249     error_counter++;
0250 else
0251     nx_packet_release(my_packet);
0252
0253 /* Close this file. */
0254 status = nx_ftp_client_file_close(&client, 100);
0255
0256 /* Disconnect from the server. */
0257 status = nx_ftp_client_disconnect(&client, 100);
0258
0259 /* Check status. */
0260 if (status)
0261     error_counter++;
0262
0263 /* Delete the FTP client. */
0264 status = nx_ftp_client_delete(&client);
0265
0266 /* Check status. */
0267 if (status)
0268     error_counter++;
0269 }
0270
0271 UINT server_login(struct NX_FTP_SERVER_STRUCT *ftp_server_ptr,
0272                  ULONG client_ip_address, UINT client_port,
0273                  CHAR *name, CHAR *password, CHAR *extra_info)
0274 {
0275     /* Always return success. */
0276     return(NX_SUCCESS);
0277 }
0278
0279 UINT server_logout(struct NX_FTP_SERVER_STRUCT *ftp_server_ptr,
0280                   ULONG client_ip_address, UINT client_port,
0281                   CHAR *name, CHAR *password, CHAR *extra_info)
0282 {
0283     /* Always return success. */
0284     return(NX_SUCCESS);
0285 }
0286
0287 }

```

Figure 1.1 Example of FTP Client and Server with NetX
(Single network interface host)

Configuration Options

There are several configuration options for building FTP for NetX. The following list describes each in detail:

Define	Meaning
NX_DISABLE_ERROR_CHECKING	Defined, this option removes the basic FTP error checking. It is typically used after the application has been debugged.
NX_FTP_SERVER_PRIORITY	The priority of the FTP Server thread. By default, this value is defined as 16 to specify priority 16.
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.
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. This define can be set by the application prior to inclusion of <i>nx_ftp.h</i> .
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. This

define can be set by the application prior to inclusion of *nx_ftp.h*.

NX_FTP_FRAGMENT_OPTION

Fragment enable for FTP TCP requests. By default, this value is `NX_DONT_FRAGMENT` to disable FTP TCP fragmenting. This define can be set by the application prior to inclusion of *nx_ftp.h*.

NX_FTP_CONTROL_WINDOW_SIZE

Control socket window size. By default, this value is 400 bytes. This define can be set by the application prior to inclusion of *nx_ftp.h*.

NX_FTP_DATA_WINDOW_SIZE

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

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

NX_FTP_SERVER_TIMEOUT

Specifies the number of ThreadX ticks that internal services will suspend for. The default value is set to 100, but can be redefined prior to inclusion of *nx_ftp.h*.

NX_FTP_USERNAME_SIZE

Specifies the number of bytes allowed in a client supplied *username*. The default value is set to 20, but can be redefined prior to inclusion of *nx_ftp.h*.

NX_FTP_PASSWORD_SIZE

Specifies the number of bytes allowed in a client supplied *password*. The default value is set to 20, but can be redefined prior to inclusion of *nx_ftp.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, but can be redefined prior to inclusion of *nx_ftp.h*.

NX_FTP_TIMEOUT_PERIOD

Specifies the number of seconds between the Server checking for client inactivity. The default value is set to 60, but can be redefined prior to inclusion of *nx_ftp.h*.

Chapter 3

Description of FTP Services

This chapter contains a description of all NetX FTP 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_ftp_client_connect`
Connect to FTP Server

`nx_ftp_client_create`
Create an FTP Client instance

`nx_ftp_client_delete`
Delete an FTP Client instance

`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

`nx_ftp_server_delete`
Delete FTP Server

`nx_ftp_server_start`
Start FTP Server

`nx_ftp_server_stop`
Stop FTP Server

nx_ftp_client_connect

Connect to an FTP Server

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 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: <div style="margin-left: 40px;"> <p>timeout value (0x00000001 through 0xFFFFFFFFE)</p> <p>TX_WAIT_FOREVER (0xFFFFFFFF)</p> <p>Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until a FTP Server responds to the request.</p> <p>Selecting a numeric value (1-0xFFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the FTP Server response.</p> </div>

Return Values

NX_SUCCESS (0x00)	Successful FTP connection.
NX_FTP_NOT_DISCONNECTED	

	(0xD4)	FTP Client is already connected
NX_FTP_200_CODE_NOT_RECEIVED		
	(0xDA)	Server rejects FTP connection
NX_FTP_300_CODE_NOT_RECEIVED		
	(0xDB)	Server rejects user/password
status		Actual NetX completion status
 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_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_PTR_ERROR	(0x16)	Invalid FTP, IP pointer, or packet pool pointer.
status		Actual NetX completion status

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`, `nx_ftp_client_delete`, `nx_ftp_client_directory_create`

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_NOT_DISCONNECTED	(0xD4)	FTP Client delete error.
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.

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`, `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: <div style="margin-left: 40px;"> timeout value (0x00000001 through 0xFFFFFFFF) 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 (1-0xFFFFFFFF) specifies the maximum number of timer-ticks to stay suspended while waiting for the FTP Server response. </div>

Return Values

NX_SUCCESS (0x00)	Successful FTP directory create.
NX_FTP_NO_2XX_RESPONSE_MKD status	Actual NetX completion status
NX_PTR_ERROR (0xE1)	FTP server error response.
	(0x16) Invalid FTP pointer.

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_connect`, `nx_ftp_client_create`,
`nx_ftp_client_directory_default_set`, `nx_ftp_client_directory_delete`,
`nx_ftp_client_directory_listing_get`, `x_ftp_client_directory_listing_continue`,

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 0xFFFFFFFF)

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 (1-0xFFFFFFFF) 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_NO_2XX_RESPONSE_CWD status	Actual NetX completion status
NX_PTR_ERROR (0xE2) (0x16)	FTP Server error response Invalid FTP pointer

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_connect`, `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_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 0xFFFFFFFF)

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 (1-0xFFFFFFFF) 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_NO_2XX_RESPONSE_RMD status		Actual NetX completion status
NX_PTR_ERROR	(0xE3) (0x16)	FTP Server error response Invalid FTP pointer

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_create`, `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_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 0xFFFFFFFF)
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 (1-0xFFFFFFFF) 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_NOT_CONNECTED	(0xD3)	FTP Client is not connected.
NX_FTP_NO_2XX_RESPONSE_PORT	(0xE4)	FTP Server response to PORT
NX_FTP_NO_1XX_RESPONSE	(0xED)	FTP Server response to NLST
status		Actual NetX completion status
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.

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_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 0xFFFFFFFF)
----------------------	---------------------------------

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 (1-0xFFFFFFFF) specifies the maximum number of timer-ticks to stay suspended while waiting for the FTP Server response.

Return Values

NX_SUCCESS	(0x00)	Successful directory listing
NX_FTP_END_OF_LISTING	(0xD8)	No more entries in directory
NX_FTP_NOT_CONNECTED	(0xD3)	FTP Client not connected
NX_FTP_NO_2XX_RESPONSE_NLST	(0xE0)	FTP server response to NLST
NX_PTR_ERROR	(0x16)	Invalid FTP pointer
Status		Actual NetX completion status

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

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 0xFFFFFFFFE)
TX_WAIT_FOREVER	(0xFFFFFFFFF)
	Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until a FTP Server responds to the request.
	Selecting a numeric value (1-0xFFFFFFFFE) 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_NOT_CONNECTED	(0xD3)	FTP Client is not connected
NX_FTP_NO_2XX_RESPONSE_DISCONNECT	(0xE5)	FTP server response to disconnect
status		Actual NetX completion status
NX_PTR_ERROR	(0x16)	Invalid FTP pointer

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_connect](#), [nx_ftp_client_create](#), [nx_ftp_client_delete](#)

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: <table data-bbox="613 982 1308 1102"> <tr> <td>timeout value</td><td>(0x00000001 through 0xFFFFFFFFE)</td></tr> <tr> <td>TX_WAIT_FOREVER</td><td>(0xFFFFFFFFF)</td></tr> </table> <p>Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until a FTP Server responds to the request.</p> <p>Selecting a numeric value (1-0xFFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the FTP Server response.</p>	timeout value	(0x00000001 through 0xFFFFFFFFE)	TX_WAIT_FOREVER	(0xFFFFFFFFF)
timeout value	(0x00000001 through 0xFFFFFFFFE)				
TX_WAIT_FOREVER	(0xFFFFFFFFF)				

Return Values

NX_SUCCESS	(0x00)	Successful FTP file close
NX_FTP_NOT_CONNECTED	(0xD3)	FTP Client is not connected
NX_FTP_NOT_OPEN	(0xD5)	
NX_FTP_NO_2XX_RESPONSE_CLOSE	(0xE6)	FTP server error response
status		Actual NetX completion status
NX_PTR_ERROR	(0x16)	Invalid FTP pointer

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_connect`, `nx_ftp_client_create`, `nx_ftp_client_delete`

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 0xFFFFFFFFE)
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 (1-0xFFFFFFFFE) 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_NOT_CONNECTED	(0xD3) FTP Client not connected
NX_FTP_NO_2XX_RESPONSE_DELE	(0xE7) FTP server response to DELE
status	Actual NetX completion status
NX_PTR_ERROR	(0x16) Invalid FTP pointer

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_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: <table data-bbox="613 1234 1308 1348"> <tr> <td>timeout value</td><td>(0x00000001 through 0xFFFFFFFF)</td></tr> <tr> <td>TX_WAIT_FOREVER</td><td>(0xFFFFFFFF)</td></tr> </table> <p>Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until a FTP Server responds to the request.</p> <p>Selecting a numeric value (1-0xFFFFFFFF) specifies the maximum number of timer-ticks to stay suspended while waiting for the FTP Server response.</p>	timeout value	(0x00000001 through 0xFFFFFFFF)	TX_WAIT_FOREVER	(0xFFFFFFFF)
timeout value	(0x00000001 through 0xFFFFFFFF)				
TX_WAIT_FOREVER	(0xFFFFFFFF)				

Return Values

NX_SUCCESS	(0x00)	Successful FTP file open
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_FTP_NO_2XX_RESPONSE_TYPE	(0xE8)	FTP Server response to TYPE
NX_FTP_NO_2XX_RESPONSE_PORT	(0xE4)	FTP Server response to PORT
NX_FTP_NO_1XX_RESPONSE	(0xED)	FTP Server that data socket not set up and connected
status		Actual NetX completion status
NX_PTR_ERROR	(0x16)	Invalid FTP pointer.

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_connect`, `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: <table> <tr> <td>timeout value</td><td>(0x00000001 through 0xFFFFFFFF)</td></tr> <tr> <td>TX_WAIT_FOREVER</td><td>(0xFFFFFFFF)</td></tr> </table> <p>Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until a FTP Server responds to the request.</p> <p>Selecting a numeric value (1-0xFFFFFFFF) specifies the maximum number of timer-ticks to stay suspended while waiting for the FTP Server response.</p>	timeout value	(0x00000001 through 0xFFFFFFFF)	TX_WAIT_FOREVER	(0xFFFFFFFF)
timeout value	(0x00000001 through 0xFFFFFFFF)				
TX_WAIT_FOREVER	(0xFFFFFFFF)				

Return Values

NX_SUCCESS	(0x00)	Successful FTP file read
NX_FTP_NOT_OPEN	(0xD5)	FTP Client is not opened
NX_FTP_END_OF_FILE	(0xD7)	End of file condition
status		Actual NetX completion status
NX_PTR_ERROR	(0x16)	Invalid FTP pointer

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_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 0xFFFFFFFF)
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 (1-0xFFFFFFFF) 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_NOT_CONNECTED	(0xD3)	FTP Client is not connected
NX_FTP_NO_3XX_RESPONSE	(0xEE)	FTP Server response to RNFR
NX_FTP_NO_2XX_RESPONSE	(0xE9)	FTP Server response to RNTD
status		Actual NetX completion status

NX_PTR_ERROR (0x16) Invalid FTP pointer.

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_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 0xFFFFFFFF)
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 (1-0xFFFFFFFF) 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_NOT_OPEN	(0xD5)	FTP Client is not opened
status		Actual NetX completion status
NX_PTR_ERROR	(0x16)	Invalid FTP pointer

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_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 <i>stack_ptr</i> .
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. 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. If this is valid, the application's login function should return NX_SUCCESS.

Return Values

NX_SUCCESS	(0x00)	Successful FTP Server create.
status		Actual NetX completion status
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_client_connect, nx_ftp_client_create, nx_ftp_client_delete,
 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_client_file_write,
 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_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_client_connect, nx_ftp_client_create, nx_ftp_client_delete,
 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_client_file_write,
 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.
status		Actual NetX completion status
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_client_connect, nx_ftp_client_create, nx_ftp_client_delete,
 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_client_file_write,
 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_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_client_connect, nx_ftp_client_create, nx_ftp_client_delete,
 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_client_file_write,
 nx_ftp_server_create, nx_ftp_server_delete, nx_ftp_server_start