



## **Trivial File Transfer Protocol (TFTP) for NetX Duo**

# **User Guide**

**Express Logic, Inc.**

858.613.6640  
Toll Free 888.THREADX  
FAX 858.521.4259

[www.expresslogic.com](http://www.expresslogic.com)

**©2002-2011 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 5.0

# Contents

---

Chapter 1 Introduction to NetX Duo TFTP .....	4
NetX Duo TFTP Requirements.....	4
TFTP File Names .....	4
TFTP Messages .....	5
TFTP Communication .....	6
TFTP Multi-Thread Support.....	6
TFTP RFCs .....	7
Chapter 2 Installation and Use of NetX Duo TFTP .....	8
Product Distribution .....	8
NetX Duo TFTP Installation.....	8
Using NetX Duo TFTP.....	8
Small Example System .....	9
Configuration Options.....	13
Chapter 3 Description of TFTP Services.....	15
nxd_tftp_client_create .....	17
nxd_tftp_client_delete .....	19
nxd_tftp_client_error_info_get.....	20
nxd_tftp_client_file_close .....	21
nx_tftp_client_file_open.....	23
nxd_tftp_client_file_open.....	25
nxd_tftp_client_file_read .....	27
nxd_tftp_client_file_write .....	30
nxd_tftp_client_packet_allocate .....	32
nxd_tftp_client_set_interface.....	34
nxd_tftp_server_create.....	36
nxd_tftp_server_delete.....	38
nxd_tftp_server_start.....	39
nxd_tftp_server_stop.....	40

# Chapter 1

## Introduction to NetX Duo TFTP

The Trivial File Transfer Protocol (TFTP) is a lightweight protocol designed for file transfers. Unlike more robust protocols, TFTP does not perform extensive error checking and can also have limited performance because it is a stop-and-wait protocol. After a TFTP data packet is sent, the sender waits for an ACK to be returned by the recipient. Although this is simple, it does limit the overall TFTP throughput. The TFTP package enables hosts to use the TFTP protocol over IP networks.

## NetX Duo TFTP Requirements

In order to function properly, the TFTP Clients portion of the NetX Duo TFTP package requires that an IP instance has already been created. In addition, UDP must be enabled on that same IP instance. The Client portion of the NetX Duo TFTP package has no further requirements.

The TFTP Server portion of the NetX Duo TFTP package has several additional requirements. First, it requires complete access to the UDP *well known port 69* for handling all client TFTP requests. The TFTP 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.

## TFTP File Names

TFTP file names should be in the format of the target file system. They should be NULL terminated ASCII strings, with full path information if necessary. There is no specified limit in the size of TFTP file names in the NetX Duo TFTP implementation.

## TFTP Messages

The TFTP has a very simple mechanism for opening, reading, writing, and closing files. There are basically 2-4 bytes of TFTP header underneath the UDP header. The definition of the TFTP file open messages has the following format:

**abcdf...f0OCTET0**

Where:

**abcd**                      2-byte Opcode field

                            0x0001 -> Open for read  
                            0x0002 -> Open for write

**f...f**                      n-byte Filename field

**0**                          1-byte NULL termination character

**OCTET**                    ASCII "OCTET" to specify binary transfer

**0**                          1-byte NULL termination character

The definition of the TFTP write, ACK, and error messages are slightly different and are defined as follows:

**abcdwxyzn...n**

Where:

**abcd**                      2-byte Opcode field

                            0x0003 -> Data packet  
                            0x0004 -> ACK for last read  
                            0x0005 -> Error condition

**wxyz**                      2-byte Block Number field (1-n)

**n...n**                      n-byte Data field

Opcode	Filename	NULL	Mode	NULL
0x0001 (read)	File Name	0	OCTET	0
0x0002 (write)	File Name	0	OCTET	0

## TFTP Communication

TFTP Servers utilize the well-known UDP port 69 to listen for Client requests. TFTP Client sockets may bind to any available UDP port. Data packet payload containing the file to upload or download is sent in 512 byte chunks, until the last packet containing < 512 bytes. Therefore a packet containing fewer than 512 bytes signals the end of file. The general sequence of events is as follows:

TFTP Read File Requests:

1. The Client issues an “Open For Read” request with the file name and waits for a reply from the Server.
2. The Server sends the first 512 bytes of the file or less if the file size is less than 512 bytes.
3. The Client receives data, sends an ACK, and waits for the next packet from the Server for files containing more than 512 bytes.
4. The sequence ends when the Client receives a packet containing fewer than 512 bytes.

TFTP Write Requests:

1. The Client issues an “Open for Write” request with the file name and waits for an ACK with a block number of 0 from the Server.
2. When the Server is ready to write the file, it sends an ACK with a block number of zero.
3. The Client sends the first 512 bytes of the file (or less for files less than 512 bytes) to the Server and waits for an ACK back.
4. The Server sends an ACK after the bytes are written.
5. The sequence ends when the Client completes writing a packet containing fewer than 512 bytes.

## TFTP Multi-Thread Support

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

## TFTP RFCs

NetX Duo TFTP is compliant with RFC1350 and related RFCs.

## Chapter 2

# Installation and Use of NetX Duo TFTP

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

## Product Distribution

NetX Duo TFTP 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:

<b><code>nxd_tftp_client.h</code></b>	Header file for NetX Duo TFTP Client
<b><code>nxd_tftp_client.c</code></b>	C Source file for NetX Duo TFTP Client
<b><code>nxd_tftp_server.h</code></b>	Header file for NetX Duo TFTP Server
<b><code>nxd_tftp_server.c</code></b>	C Source file for NetX Duo TFTP Server
<b><code>filex_stub.h</code></b>	Stub file if FileX is not present
<b><code>nxd_tftp.pdf</code></b>	PDF description of NetX Duo TFTP
<b><code>demo_netxduo_tftp.c</code></b>	NetX Duo TFTP demonstration

## NetX Duo TFTP Installation

To use NetX Duo TFTP, the entire distribution mentioned previously may 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\_tftp\_client.h*, *nxd\_tftp\_client.c*, *nxd\_tftp\_server.h* and *nxd\_tftp\_server.c* files could be copied into this directory.

## Using NetX Duo TFTP

Using NetX Duo TFTP is easy. Basically, the application code must include *nxd\_tftp\_client.h* and/or *nxd\_tftp\_server.h* after it includes *tx\_api.h*, *fx\_api.h*, and *nx\_api.h*, in order to use ThreadX, FileX, and NetX Duo, respectively. The application project must also include *nxd\_tftp\_client.c* and/or *nxd\_tftp\_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 Duo TFTP. Once the header file(s) is included, the application code is then able to use TFTP services.

Note that since TFTP utilizes NetX Duo UDP services, UDP must be enabled with the *nx\_udp\_enable* call prior to using TFTP.

## Small Example System

An example of how easy it is to use NetX Duo TFTP is described in Figure 1.1 that appears below. In this example, the TFTP include file *nxd\_tftp\_client.h* and *nxd\_tftp\_server.h* are brought in at line 4. Next, the TFTP Server is created in “*tx\_application\_define*” at line 120. Note that the TFTP Server control block “*server*” was defined as a global variable at line 22 previously. This demo chooses to use IPv4 for its TFTP communication in line 34. After successful creation, the TFTP Server is started at line 129. At line 165 the TFTP Client is created. And finally, the Client writes the file at line 196 and reads the file back at line 218.

Note that this example uses FileX for the TFTP Server handling of receiving and downloading TFTP Client file requests. However, if *NX\_TFTP\_NO\_FILEX* is defined, the application can include *file\_stub.h* instead of *fx\_api.h*.

Also note that existing NetX TFTP client and server applications will work with NetX Duo TFTP. However, the application developer is encouraged to port their Netx TFTP applications to NetX Duo. The equivalent NetX TFTP services are:

```
nxd_tftp_server_start
nxd_tftp_server_stop
nxd_tftp_client_file_read
nxd_tftp_client_file_write
nxd_tftp_client_file_open
```

```
0001 #include "tx_api.h"
0002 #include "nx_api.h"
0003 #include "fx_api.h"
0004 #include "nxd_tftp_client.h"
0005 #include "nxd_tftp_server.h"
0006
0007 #define DEMO_STACK_SIZE 2048
0008
0009 /* Define the ThreadX, NetX, and FileX object control blocks... */
0010
0011 TX_THREAD client_thread;
0012 NX_PACKET_POOL server_pool;
0013 NX_IP server_ip;
0014 NX_PACKET_POOL client_pool;
0015 NX_IP client_ip;
0016 FX_MEDIA ram_disk;
0017
0018
0019 /* Define the NetX TFTP object control blocks. */
0020
```

```

0021 NX_TFTP_CLIENT      client;
0022 NX_TFTP_SERVER      server;
0023
0024
0025 /* Define the counters used in the demo application... */
0026 ULONG                  error_counter;
0027
0028 /* Define the memory area for the FileX RAM disk. */
0029 UCHAR                  ram_disk_memory[32000];
0030
0031/* Define which IP protocol to use for TFTP, IPV4 or IPV6. To use
0032 IPv6, NetX Duo must be enabled with IPV6. */
0033 #define IP_TYPE         4
0034
0035 /* Define function prototypes. */
0036
0037 VOID    _fx_ram_driver(FX_MEDIA *media_ptr);
0038 VOID    _nx_ram_network_driver(NX_IP_DRIVER *driver_req_ptr);
0039 void     client_thread_entry(ULONG thread_input);
0040
0041
0042 /* Define main entry point. */
0043
0044 int main()
0045 {
0046
0047     /* Enter the ThreadX kernel. */
0048     tx_kernel_enter();
0049 }
0050
0051
0052 /* Define what the initial system looks like. */
0053
0054 void     tx_application_define(void *first_unused_memory)
0055 {
0056
0057     UINT     status;
0058     UCHAR    *pointer;
0059
0060
0061     /* Setup the working pointer. */
0062     pointer = (UCHAR *) first_unused_memory;
0063
0064     /* Create the main TFTP demo thread. */
0065     status = tx_thread_create(&client_thread, "thread 0",
0066                             client_thread_entry, 0, pointer, DEMO_STACK_SIZE,
0067                             6, 6, TX_NO_TIME_SLICE, TX_AUTO_START);
0068     pointer += DEMO_STACK_SIZE ;
0069
0070     /* Check for errors. */
0071     if (status)
0072         error_counter++;
0073
0074     /* Open the RAM disk. */
0075     status = fx_media_open(&ram_disk, "RAM DISK", _fx_ram_driver,
0076                           ram_disk_memory, pointer, 4096);
0077     pointer += 4096 ;
0078
0079     /* Check for errors. */
0080     if (status)
0081         error_counter++;
0082
0083     /* Create the packet pool for the TFTP Server. The packet size must
0084        be a minimum of 560 bytes. */
0085     status = nx_packet_pool_create(&server_pool, "NetX Server Packet Pool",
0086                                   560, pointer, 8192);
0087     pointer = pointer + 8192;
0088
0089     /* Check for errors. */
0090     if (status)
0091         error_counter++;
0092
0093     /* Create the IP instance for the TFTP Server. */
0094     status = nx_ip_create(&server_ip, "NetX Server IP Instance",
0095                           IP_ADDRESS(1,2,3,4), 0xFFFFFFFF0UL,
0096                           &server_pool, _nx_ram_network_driver,
0097                           pointer, 2048, 1);
0098     pointer = pointer + 2048;
0099
0100     /* Check for errors. */
0101     if (status)

```



```

0182
0183 /* Check status. */
0184 if (status)
0185     error_counter++;
0186
0187 /* Write ABCs into the packet payload! */
0188 memcpy(my_packet -> nx_packet_prepend_ptr,
0189        "ABCDEFGHIIJKLMNOPQRSTUVWXYZ ", 28);
0190
0191 /* Adjust the write pointer. */
0192 my_packet -> nx_packet_length = 28;
0193 my_packet -> nx_packet_append_ptr = my_packet -> nx_packet_prepend_ptr+28;
0194
0195 /* Write this packet to the file via TFTP. */
0196 status = nxd_tftp_client_file_write(&client, my_packet, 300, IP_TYPE);
0197
0198 /* Check status. */
0199 if (status)
0200     error_counter++;
0201
0202 /* Close this file. */
0203 status = nxd_tftp_client_file_close(&client, IP_TYPE);
0204
0205 /* Check status. */
0206 if (status)
0207     error_counter++;
0208
0209 /* Open the same file for reading. */
0210 status = nxd_tftp_client_file_open(&client, "test.txt",
0211                                   IP_ADDRESS(1,2,3,4), NX_TFTP_OPEN_FOR_READ, 200,
0212                                   IP_TYPE);
0213
0214 /* Check status. */
0215 if (status)
0216     error_counter++;
0217
0218 /* Read the file back. */
0219 status = nxd_tftp_client_file_read(&client, &my_packet, 200, IP_TYPE);
0220
0221 /* Check status. */
0222 if (status)
0223     error_counter++;
0224 else
0225     nx_packet_release(my_packet);
0226
0227 /* Close the file again. */
0228 status = nxd_tftp_client_file_close(&client) , IP_TYPE);
0229
0230 /* Check status. */
0231 if (status)
0232     error_counter++;
0233
0234 /* Delete the client. */
0235 status = nxd_tftp_client_delete(&client);
0236
0237 /* Check status. */
0238 if (status)
0239     error_counter++;
0240 }

```

Figure 1.1 Example of TFTP use with NetX Duo

## Configuration Options

There are several configuration options for building NetX Duo TFTP. The following list describes each in detail. Unless otherwise specified, these options are found in *nxd\_tftp\_client.h* and *nxd\_tftp\_server.h*.

Define	Meaning
<b>NX_DISABLE_ERROR_CHECKING</b>	This option removes the basic TFTP error checking. It is typically used after the application has been debugged. By default it is not defined.
<b>NX_TFTP_SERVER_PRIORITY</b>	The priority of the TFTP server thread. The default value is 16 to specify priority 16 in <i>nxd_tftp_server.h</i> .
<b>NX_TFTP_MAX_CLIENTS</b>	The maximum number of clients the server can handle at one time. The default value is 10 in <i>nxd_tftp_server.h</i> .
<b>NX_TFTP_ERROR_STRING_MAX</b>	The maximum number of characters in the error string. By default, this value is 64.
<b>NX_TFTP_NO_FILEX</b>	Defined, this option provides a stub for FileX dependencies. The TFTP Client will function without any change if this option is defined. The TFTP Server will need to either be modified or the user will have to create a handful of FileX services in order to function properly.
<b>NX_TFTP_TYPE_OF_SERVICE</b>	Type of service required for the TFTP requests. By default, this value is defined as <code>NX_IP_NORMAL</code> to indicate normal IP packet service.
<b>NX_TFTP_FRAGMENT_OPTION</b>	This option enables

fragmentation of the TFTP packet. By default, this value is `NX_DONT_FRAGMENT` to disable fragmentation.

#### **`NX_TFTP_TIME_TO_LIVE`**

This option specifies the number of routers the TFTP packet can pass before it is discarded. The default value is set to 0x80.

#### **`NX_TFTP_SOURCE_PORT`**

This option allows the TFTP Client host application to specify the TFTP Client UDP socket port. It is defaulted to `NX_ANY_PORT` in *nxd\_tftp\_client.h*.

## Chapter 3

# Description of TFTP Services

This chapter contains a description of all NetX Duo TFTP services (listed below) in alphabetic order. Unless otherwise specified, all services support IPv6 and IPv4 communications.

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_tftp_client_file_open`  
*Open TFTP client file (IPv4 only)*

`nxd_tftp_client_file_open`  
*Open TFTP client file*

`nxd_tftp_client_create`  
*Create a TFTP client instance*

`nxd_tftp_client_delete`  
*Delete a TFTP client instance*

`nxd_tftp_client_error_info_get`  
*Get client error information*

`nxd_tftp_client_file_close`  
*Close client file*

`nxd_tftp_client_file_open`  
*Open client file*

`nxd_tftp_client_file_read`  
*Read a block from client file*

`nxd_tftp_client_file_write`  
*Write block to client file*

`nxd_tftp_client_packet_allocate`  
*Allocate packet for client file write*

nxd\_tftp\_client\_packet\_allocate  
*Set the physical interface for TFTP requests*

nxd\_tftp\_server\_create  
*Create TFTP server*

nxd\_tftp\_server\_delete  
*Delete TFTP server*

nxd\_tftp\_server\_start  
*Start TFTP server*

nxd\_tftp\_server\_stop  
*Stop TFTP server*



# nxd\_tftp\_client\_create

Create a TFTP Client instance

## Prototype

```
UINT nxd_tftp_client_create(NX_TFTP_CLIENT *tftp_client_ptr,
    CHAR *tftp_client_name, NX_IP *ip_ptr, NX_PACKET_POOL *pool_ptr);
```

## Description

This service creates a TFTP Client instance for the previously created IP instance.

**Important Note:** The application must make certain the supplied IP and packet pool are already created. In addition, UDP must be enabled for the IP instance prior to calling this service.

## Input Parameters

<b>tftp_client_ptr</b>	Pointer to TFTP Client control block.
<b>tftp_client_name</b>	Name of this TFTP Client instance
<b>ip_ptr</b>	Pointer to previously created IP instance.
<b>pool_ptr</b>	Pointer to packet pool TFTP Client instance.

## Return Values

<b>NX_SUCCESS</b>	(0x00)	Successful TFTP create.
<b>status</b>		Actual NetX completion status
<b>NX_PTR_ERROR</b>	(0x16)	Invalid IP, pool, or TFTP pointer.

## Allowed From

Initialization and Threads

## Example

```
/* Create a TFTP Client instance. */
status = nxd_tftp_client_create(&my_tftp_client, "My TFTP Client",
    &my_ip, &pool_ptr);

/* If status is NX_SUCCESS a TFTP Client instance was successfully
   created. */
```

**See Also**

nxd\_tftp\_client\_delete, nxd\_tftp\_client\_error\_info\_get,  
nxd\_tftp\_client\_file\_close, nxd\_tftp\_client\_file\_open,  
nxd\_tftp\_client\_file\_read, nxd\_tftp\_client\_file\_write,  
nxd\_tftp\_client\_packet\_allocate, nxd\_tftp\_server\_create,  
nxd\_tftp\_server\_delete, nxd\_tftp\_server\_start, nxd\_tftp\_server\_stop

# nxd\_tftp\_client\_delete

Delete a TFTP Client instance

## Prototype

```
UINT nxd_tftp_client_delete(NX_TFTP_CLIENT *tftp_client_ptr);
```

## Description

This service deletes a previously created TFTP Client instance.

## Input Parameters

<b>tftp_client_ptr</b>	Pointer to previously created TFTP client instance.
------------------------	---

## Return Values

<b>NX_SUCCESS</b>	(0x00)	Successful TFTP Client delete.
<b>NX_PTR_ERROR</b>	(0x16)	Invalid pointer input.
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of this service.

## Allowed From

Threads

## Example

```
/* Delete a TFTP Client instance. */
status = nxd_tftp_client_delete(&my_tftp_client);

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

## See Also

nxd\_tftp\_client\_create, nxd\_tftp\_client\_error\_info\_get,  
 nxd\_tftp\_client\_file\_close, nxd\_tftp\_client\_file\_open,  
 nxd\_tftp\_client\_file\_read, nxd\_tftp\_client\_file\_write,  
 nxd\_tftp\_client\_packet\_allocate, nxd\_tftp\_server\_create,  
 nxd\_tftp\_server\_delete, nxd\_tftp\_server\_start, nxd\_tftp\_server\_stop

## **nxd\_tftp\_client\_error\_info\_get**

Get client error information

### **Prototype**

```
UINT nxd_tftp_client_error_info_get(NX_TFTP_CLIENT *tftp_client_ptr,
                                     UINT *error_code, CHAR **error_string);
```

### **Description**

This service returns the last error code received and sets the pointer to the client's internal error string. In error conditions, the user can view the last error sent by the server. A null error string indicates no error is present.

### **Input Parameters**

<b>tftp_client_ptr</b>	Pointer to previously created TFTP Client instance.
<b>error_code</b>	Pointer to destination area for error code
<b>error_string</b>	Pointer to destination for error string

### **Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successful TFTP error info get.
<b>NX_PTR_ERROR</b>	(0x16)	Invalid TFTP Client pointer.
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of this service.

### **Allowed From**

Threads

### **Example**

```
/* Get error information for client. */
status = nxd_tftp_client_error_info_get(&my_tftp_client, &error_code,
                                         &error_string_ptr);

/* If status is NX_SUCCESS the error code and error string are available. */
```

### **See Also**

nxd\_tftp\_client\_create, nxd\_tftp\_client\_delete, nxd\_tftp\_client\_file\_close,  
 nxd\_tftp\_client\_file\_open, nxd\_tftp\_client\_file\_read,  
 nxd\_tftp\_client\_file\_write, nxd\_tftp\_client\_packet\_allocate,  
 nxd\_tftp\_server\_create, nxd\_tftp\_server\_delete, nxd\_tftp\_server\_start,  
 nxd\_tftp\_server\_stop

## **nxd\_tftp\_client\_file\_close**

Close client file

### **Prototype**

```
UINT nxd_tftp_client_file_close(NX_TFTP_CLIENT *tftp_client_ptr,
                               UINT ip_type);
```

### **Description**

This service closes the previously opened file by this TFTP Client instance. A TFTP Client instance is allowed to have only one file open at a time.

### **Input Parameters**

<b>tftp_client_ptr</b>	Pointer to previously created TFTP Client instance.
<b>ip_type</b>	Indicate which IP protocol to use. Valid options are IPv4 (4) or IPv6 (6).

### **Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successful TFTP file close.
<b>status</b>		Actual NetX completion status
<b>NX_PTR_ERROR</b>	(0x16)	Invalid pointer input.
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of this service.

### **Allowed From**

Threads

### **Example**

```
/* Close the previously opened file associated with "my_client". */
status = nxd_tftp_client_file_close(&my_tftp_client);

/* If status is NX_SUCCESS the TFTP file is closed. */
```

### **See Also**

nxd\_tftp\_client\_create, nxd\_tftp\_client\_delete,  
 nxd\_tftp\_client\_error\_info\_get, nxd\_tftp\_client\_file\_open,  
 nxd\_tftp\_client\_file\_read, nxd\_tftp\_client\_file\_write,  
 nxd\_tftp\_client\_packet\_allocate, nxd\_tftp\_server\_create,  
 nxd\_tftp\_server\_delete, nxd\_tftp\_server\_start, nxd\_tftp\_server\_stop



## **nx\_tftp\_client\_file\_open**

Open TFTP client file

### **Prototype**

```
UINT nx_tftp_client_file_open(NX_TFTP_CLIENT *tftp_client_ptr,
                              CHAR *file_name, NXD_ADDRESS *server_ip_address, UINT
                              open_type, ULONG wait_option);
```

### **Description**

This service attempts to open the specified file on the TFTP Server at the specified IP address. The file will be opened for either reading or writing. Note this is limited to IPv4 packets only, and is intended for supporting NetX TFTP applications. Developers are encouraged to port their applications to using equivalent “duo” service *nxd\_tftp\_client\_file\_open*.

### **Input Parameters**

**tftp\_client\_ptr** Pointer to TFTP control block.

**file\_name** ASCII file name, NULL-terminated and with appropriate path information.

**server\_ip\_address** Server TFTP address.

**open\_type** Type of open request, either:

**NX\_TFTP\_OPEN\_FOR\_READ** (0x01)

**NX\_TFTP\_OPEN\_FOR\_WRITE** (0x02)

**wait\_option** Defines how long the service will wait for the TFTP Client file open. 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 TFTP 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 TFTP server response.

**ip\_type**                      Indicate which IP protocol to use. Valid options are IPv4 (4) or IPv6 (6).

## Return Values

<b>NX_SUCCESS</b>	(0x00)	Successful Client file open
<b>NX_TFTP_NOT_CLOSED</b>	(0xC3)	Client already has file open
<b>NX_INVALID_TFTP_SERVER_ADDRESS</b>	(0x08)	Invalid server address received
<b>NX_TFTP_NO_ACK_RECEIVED</b>	(0x09)	No ACK received from server
<b>NX_PTR_ERROR</b>	(0x16)	Invalid pointer input.
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of this service
<b>NX_IP_ADDRESS_ERROR</b>	(0x21)	Invalid Server IP address
<b>NX_OPTION_ERROR</b>	(0x0a)	Invalid open type

## Allowed From

Threads

## Example

```
/* Define the TFTP server address. */
NXD_ADDRESS server_ip_address;

server_ip_address.nxd_ip_version = NX_IP_VERSION_V6;
server_ip_address.nxd_ip_address.v6[0] = 0x20010db8;
server_ip_address.nxd_ip_address.v6[1] = 0xf101;
server_ip_address.nxd_ip_address.v6[2] = 0;
server_ip_address.nxd_ip_address.v6[3] = 0x101;

/* Open file "test.txt" for reading on the TFTP Server. */
status = nxd_tftp_client_file_open(&my_tftp_client, "test.txt",
                                   &server_ip_address, NX_TFTP_OPEN_FOR_READ, 200);

/* If status is NX_SUCCESS the "test.txt" file is now open for reading. */
```

## See Also

nxd\_tftp\_client\_create, nxd\_tftp\_client\_delete,  
 nxd\_tftp\_client\_error\_info\_get, nxd\_tftp\_client\_file\_close,  
 nxd\_tftp\_client\_file\_read, nxd\_tftp\_client\_file\_write,  
 nxd\_tftp\_client\_packet\_allocate, nxd\_tftp\_server\_create,  
 nxd\_tftp\_server\_delete, nxd\_tftp\_server\_start, nxd\_tftp\_server\_stop



## **nxd\_tftp\_client\_file\_open**

Open TFTP client file

### **Prototype**

```
UINT nxd_tftp_client_file_open(NX_TFTP_CLIENT *tftp_client_ptr,
                               CHAR *file_name, NXD_ADDRESS *server_ip_address, UINT
                               open_type, ULONG wait_option, UINT ip_type);
```

### **Description**

This service attempts to open the specified file on the TFTP Server at the specified IPv6 address. The file will be opened for either reading or writing.

### **Input Parameters**

**tftp\_client\_ptr** Pointer to TFTP control block.

**file\_name** ASCII file name, NULL-terminated and with appropriate path information.

**server\_ip\_address** Server TFTP address.

**open\_type** Type of open request, either:

**NX\_TFTP\_OPEN\_FOR\_READ** (0x01)  
**NX\_TFTP\_OPEN\_FOR\_WRITE** (0x02)

**wait\_option** Defines how long the service will wait for the TFTP Client file open. 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 TFTP 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 TFTP server response.

**ip\_type** Indicate which IP protocol to use. Valid options

are IPv4 (4) or IPv6 (6).

## Return Values

<b>NX_SUCCESS</b>	(0x00)	Successful Client file open
<b>NX_TFTP_NOT_CLOSED</b>	(0xC3)	Client already has file open
<b>NX_INVALID_TFTP_SERVER_ADDRESS</b>	(0x08)	Invalid server address received
<b>NX_TFTP_NO_ACK_RECEIVED</b>	(0x09)	No ACK received from server
<b>status</b>		Actual completion status
<b>NX_PTR_ERROR</b>	(0x16)	Invalid pointer input.
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of this service
<b>NX_IP_ADDRESS_ERROR</b>	(0x21)	Invalid Server IP address
<b>NX_OPTION_ERROR</b>	(0x0a)	Invalid open type

## Allowed From

Threads

## Example

```
/* Define the TFTP server address. */
NXD_ADDRESS server_ip_address;

server_ip_address.nxd_ip_version = NX_IP_VERSION_V6;
server_ip_address.nxd_ip_address.v6[0] = 0x20010db8;
server_ip_address.nxd_ip_address.v6[1] = 0xf101;
server_ip_address.nxd_ip_address.v6[2] = 0;
server_ip_address.nxd_ip_address.v6[3] = 0x101;

/* Open file "test.txt" for reading on the TFTP Server. */
status = nxd_tftp_client_file_open(&my_tftp_client, "test.txt",
                                   &server_ip_address, NX_TFTP_OPEN_FOR_READ, 200);

/* If status is NX_SUCCESS the "test.txt" file is now open for reading. */
```

## See Also

nxd\_tftp\_client\_create, nxd\_tftp\_client\_delete,  
 nxd\_tftp\_client\_error\_info\_get, nxd\_tftp\_client\_file\_close,  
 nxd\_tftp\_client\_file\_read, nxd\_tftp\_client\_file\_write,  
 nxd\_tftp\_client\_packet\_allocate, nxd\_tftp\_server\_create,  
 nxd\_tftp\_server\_delete, nxd\_tftp\_server\_start, nxd\_tftp\_server\_stop

## **nxd\_tftp\_client\_file\_read**

Read a block from client file

### **Prototype**

```
UINT nxd_tftp_client_file_read(NX_TFTP_CLIENT *tftp_client_ptr,
                               NX_PACKET **packet_ptr, ULONG wait_option,
                               UINT ip_type);
```

### **Description**

This service reads a 512-byte block from the previously opened TFTP Client file. A block containing fewer than 512 bytes signals the end of the file.

### **Input Parameters**

<b>tftp_client_ptr</b>	Pointer to TFTP Client control block.
<b>packet_ptr</b>	Destination for packet containing the block read from the file.
<b>wait_option</b>	Defines how long the service will wait for the read to complete. The wait options are defined as follows:  <b>timeout value</b> (0x00000001 through 0xFFFFFFFF) <b>TX_WAIT_FOREVER</b> (0xFFFFFFFF)  Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the TFTP 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 TFTP server to send a block of the file.
<b>ip_type</b>	Indicate which IP protocol to use. Valid options are IPv4 (4) or IPv6 (6).

### **Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successful Client block read
-------------------	--------	------------------------------

**NX\_TFTP\_NOT\_OPEN**

(0xC3)

Specified Client file  
is not open for reading

<b>NX_NO_PACKET</b>	(0x01)	No Packet received from Server.
<b>NX_INVALID_TFTP_SERVER_ADDRESS</b>	(0x08)	Invalid server address received
<b>NX_TFTP_NO_ACK_RECEIVED</b>	(0x09)	No ACK received from Server
<b>status</b>		Actual completion status
<b>NX_TFTP_END_OF_FILE</b>	(0xC5)	End of file detected.
<b>NX_PTR_ERROR</b>	(0x16)	Invalid pointer input.
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of this service

### Allowed From

Threads

### Example

```
/* Read a block from a previously opened file of "my_client". */
status = nxd_tftp_client_file_read(&my_tftp_client, &return_packet_ptr, 200);

/* If status is NX_SUCCESS a block of the TFTP file is in the payload of
   "return_packet_ptr". */
```

### See Also

nxd\_tftp\_client\_create, nxd\_tftp\_client\_delete,  
nxd\_tftp\_client\_error\_info\_get, nxd\_tftp\_client\_file\_close,  
nxd\_tftp\_client\_file\_open, nxd\_tftp\_client\_file\_write,  
nxd\_tftp\_client\_packet\_allocate, nxd\_tftp\_server\_create,  
nxd\_tftp\_server\_delete, nxd\_tftp\_server\_start, nxd\_tftp\_server\_stop

# nxd\_tftp\_client\_file\_write

Write a block to Client file

## Prototype

```
UINT nxd_tftp_client_file_write(NX_TFTP_CLIENT *tftp_client_ptr,
                                NX_PACKET *packet_ptr, ULONG wait_option, UINT ip_type);
```

## Description

This service writes a 512-byte block to the previously opened TFTP Client file. Specifying a block containing fewer than 512 bytes signals the end of the file.

## Input Parameters

<b>tftp_client_ptr</b>	Pointer to TFTP Client control block.
<b>packet_ptr</b>	Packet containing the block to write to the file.
<b>wait_option</b>	Defines how long the service will wait for the write to complete. The wait options are defined as follows:  <b>timeout value</b> (0x00000001 through 0xFFFFFFFF) <b>TX_WAIT_FOREVER</b> (0xFFFFFFFF)  Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the TFTP 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 TFTP server to send an ACK for the write request.
<b>ip_type</b>	Indicate which IP protocol to use. Valid options are IPv4 (4) or IPv6 (6).

## Return Values

<b>NX_SUCCESS</b>	(0x00)	Successful Client block write
<b>NX_TFTP_NOT_OPEN</b>	(0xC3)	Specified Client file is not open for writing

<b>NX_TFTP_TIMEOUT</b>	(0xC1)	Timeout waiting for Server ACK
<b>NX_INVALID_TFTP_SERVER_ADDRESS</b>	<b>(0x08)</b>	Invalid server address received
<b>NX_TFTP_NO_ACK_RECEIVED</b>	<b>(0x09)</b>	No ACK received from server
<b>status</b>		Actual completion status
<b>NX_PTR_ERROR</b>	(0x16)	Invalid pointer input.
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of this service

## Allowed From

Threads

## Example

```
/* Write a block to the previously opened file of "my_client". */
status = nxd_tftp_client_file_write(&my_tftp_client, packet_ptr, 200);

/* If status is NX_SUCCESS the block in the payload of "packet_ptr" was
   written to the TFTP file opened by "my_client". */
```

## See Also

nxd\_tftp\_client\_create, nxd\_tftp\_client\_delete,  
 nxd\_tftp\_client\_error\_info\_get, nxd\_tftp\_client\_file\_close,  
 nxd\_tftp\_client\_file\_open, nxd\_tftp\_client\_file\_read,  
 nxd\_tftp\_client\_packet\_allocate, nxd\_tftp\_server\_create,  
 nxd\_tftp\_server\_delete, nxd\_tftp\_server\_start, nxd\_tftp\_server\_stop

# nxd\_tftp\_client\_packet\_allocate

Allocate packet for Client file write

## Prototype

```
UINT nxd_tftp_client_packet_allocate(NX_PACKET_POOL *pool_ptr,
                                     NX_PACKET **packet_ptr, ULONG wait_option,
                                     UINT ip_type)
```

## Description

This service allocates a UDP packet from the specified packet pool and makes room for the 4-byte TFTP header before the packet is returned to the caller. The caller can then build a buffer for writing to a client file.

## Input Parameters

<b>pool_ptr</b>	Pointer to packet pool.
<b>packet_ptr</b>	Destination for pointer to allocated packet.
<b>wait_option</b>	Defines how long the service will wait for the packet allocate to complete. The wait options are defined as follows:  <b>timeout value</b> (0x00000001 through 0xFFFFFFFF) <b>TX_WAIT_FOREVER</b> (0xFFFFFFFF)  Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the allocation completes.  Selecting a numeric value (1-0xFFFFFFFF) specifies the maximum number of timer-ticks to stay suspended while waiting for the packet allocation.
<b>ip_type</b>	Indicate which IP protocol to use. Valid options are IPv4 (4) or IPv6 (6).

## Return Values

<b>NX_SUCCESS</b>	(0x00)	Successful packet allocate
-------------------	--------	----------------------------



<b>NX_NO_PACKET</b>	(0x01)	No packet available
<b>status</b>		Actual NetX completion status
NX_PTR_ERROR	(0x16)	Invalid pointer input.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

### Allowed From

Threads

### Example

```
/* Allocate a packet for TFTP file write. */
status = nxd_tftp_client_packet_allocate(&my_pool, &packet_ptr, 200);
/* If status is NX_SUCCESS "packet_ptr" contains the new packet. */
```

### See Also

nxd\_tftp\_client\_create, nxd\_tftp\_client\_delete,  
nxd\_tftp\_client\_error\_info\_get, nxd\_tftp\_client\_file\_close,  
nxd\_tftp\_client\_file\_open, nxd\_tftp\_client\_file\_read,  
nxd\_tftp\_client\_file\_write, nxd\_tftp\_server\_create, nxd\_tftp\_server\_delete,  
nxd\_tftp\_server\_start, nxd\_tftp\_server\_stop

# nxd\_tftp\_client\_set\_interface

---

Set physical interface for TFTP requests

**Prototype**

```
UINT nxd_tftp_client_set_interface(NX_TFTP_CLIENT *tftp_client_ptr,
                                   UINT if_index)
```

**Description**

This service uses the input interface index to set the physical interface for the TFTP Client to send and receive TFTP packets. The default value is zero, for the primary interface. Note that NetX Duo must support multihome addressing (v5.6 or later) to use this service.

**Input Parameters**

<b>tftp_client_ptr</b>	Pointer to TFTP Client instance
<b>if_index</b>	Index of physical interface to use

**Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successfully set interface
<b>NX_TFTP_INVALID_INTERFACE</b>	(0x0B)	Invalid interface input
<b>NX_PTR_ERROR</b>	(0x16)	Invalid pointer input.
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of this service

**Allowed From**

Threads

**Example**

```
/* Specify the primary interface for TFTP requests. */
status = nxd_tftp_client_set_interface(&client, 0);

/* If status is NX_SUCCESS the primary interface will be use for TFTP
communications. */
```

**See Also**

nxd\_tftp\_client\_create, nxd\_tftp\_client\_delete,  
nxd\_tftp\_client\_error\_info\_get, nxd\_tftp\_client\_file\_close,

nxd\_tftp\_client\_file\_open, nxd\_tftp\_client\_file\_read,  
nxd\_tftp\_client\_file\_write, nxd\_tftp\_server\_create, nxd\_tftp\_server\_delete,  
nxd\_tftp\_server\_start, nxd\_tftp\_server\_stop

# nxd\_tftp\_server\_create

Create TFTP server

## Prototype

```
UINT nxd_tftp_server_create(NX_TFTP_SERVER *tftp_server_ptr,
    CHAR *tftp_server_name, NX_IP *ip_ptr, FX_MEDIA *media_ptr,
    VOID *stack_ptr, ULONG stack_size,
    NX_PACKET_POOL *pool_ptr);
```

## Description

This service creates a TFTP Server that responds to TFTP Client requests on port 69. The Server must be started by a subsequent call to *nxd\_tftp\_server\_start*.

**Important Note:** The application must make certain the supplied IP instance, packet pool, and FileX media instance are already created. In addition, UDP must be enabled for the IP instance prior to calling this service.

## Input Parameters

<b>tftp_server_ptr</b>	Pointer to TFTP Server control block.
<b>tftp_server_name</b>	Name of this TFTP Server instance
<b>ip_ptr</b>	Pointer to previously created IP instance.
<b>media_ptr</b>	Pointer to FileX media instance.
<b>stack_ptr</b>	Pointer to TFTP Server stack area.
<b>stack_size</b>	Number of bytes in the TFTP Server stack.
<b>pool_ptr</b>	Pointer to TFTP packet pool. Note that the supplied pool must have packet payloads at least 580 bytes in size. <sup>1</sup>

## Return Values

<b>NX_SUCCESS</b>	(0x00)	Successful Server create
-------------------	--------	--------------------------

<sup>1</sup> The data portion of a packet is exactly 512 bytes, but the packet payload size must be at least 572 bytes. The remaining bytes are used for the UDP, IPv6, and Ethernet headers and potential trailing bytes required by the driver for alignment.

<b>NX_TFTP_POOL_ERROR</b>	(0xC6)	Packet pool has packet size of less than 560 bytes
<b>status</b>		Actual completion status
<b>NX_PTR_ERROR</b>	(0x16)	Invalid pointer input.

### Allowed From

Initialization, Threads

### Example

```
/* Create a TFTP Server called "my_server". */
status = nxd_tftp_server_create(&my_server, "My TFTP Server", &server_ip,
                                &ram_disk, stack_ptr, 2048, pool_ptr);

/* If status is NX_SUCCESS the TFTP Server is created. */
```

### See Also

nxd\_tftp\_client\_create, nxd\_tftp\_client\_delete,  
nxd\_tftp\_client\_error\_info\_get, nxd\_tftp\_client\_file\_close,  
nxd\_tftp\_client\_file\_open, nxd\_tftp\_client\_file\_read,  
nxd\_tftp\_client\_file\_write, nxd\_tftp\_client\_packet\_allocate,  
nxd\_tftp\_server\_delete, nxd\_tftp\_server\_start, nxd\_tftp\_server\_stop

## nxd\_tftp\_server\_delete

---

Delete TFTP Server

### Prototype

```
UINT nxd_tftp_server_delete(NX_TFTP_SERVER *tftp_server_ptr);
```

### Description

This service deletes a previously created TFTP Server.

### Input Parameters

<b>tftp_server_ptr</b>	Pointer to TFTP Server control block.
------------------------	---------------------------------------

### Return Values

<b>NX_SUCCESS</b>	(0x00)	Successful Server delete
<b>NX_PTR_ERROR</b>	(0x16)	Invalid pointer input.
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of this service

### Allowed From

Threads

### Example

```
/* Delete the TFTP Server called "my_server". */
status = nxd_tftp_server_delete(&my_server);

/* If status is NX_SUCCESS the TFTP Server is deleted. */
```

### See Also

nxd\_tftp\_client\_create, nxd\_tftp\_client\_delete,  
 nxd\_tftp\_client\_error\_info\_get, nxd\_tftp\_client\_file\_close,  
 nxd\_tftp\_client\_file\_open, nxd\_tftp\_client\_file\_read,  
 nxd\_tftp\_client\_file\_write, nxd\_tftp\_client\_packet\_allocate,  
 nxd\_tftp\_server\_create, nxd\_tftp\_server\_start, nxd\_tftp\_server\_stop

## **nxd\_tftp\_server\_start**

---

Start TFTP server

### **Prototype**

```
UINT nxd_tftp_server_start(NX_TFTP_SERVER *tftp_server_ptr);
```

### **Description**

This service starts the previously created TFTP Server.

### **Input Parameters**

<b>tftp_server_ptr</b>	Pointer to TFTP Server control block.
------------------------	---------------------------------------

### **Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successful Server start
<b>NX_PTR_ERROR</b>	(0x16)	Invalid pointer input

### **Allowed From**

Initialization, threads

### **Example**

```
/* Start the TFTP Server called "my_server". */
status = nxd_tftp_server_start(&my_server);

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

### **See Also**

nxd\_tftp\_client\_create, nxd\_tftp\_client\_delete,  
 nxd\_tftp\_client\_error\_info\_get, nxd\_tftp\_client\_file\_close,  
 nxd\_tftp\_client\_file\_open, nxd\_tftp\_client\_file\_read,  
 nxd\_tftp\_client\_file\_write, nxd\_tftp\_client\_packet\_allocate,  
 nxd\_tftp\_server\_create, nxd\_tftp\_server\_delete, nxd\_tftp\_server\_stop

## **nxd\_tftp\_server\_stop**

Stop TFTP Server

### **Prototype**

```
UINT nxd_tftp_server_stop(NX_TFTP_SERVER *tftp_server_ptr);
```

### **Description**

This service stops the previously created TFTP Server.

### **Input Parameters**

**tftp\_server\_ptr**                      Pointer to TFTP Server control block.

### **Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successful Server stop
<b>NX_PTR_ERROR</b>	(0x16)	Invalid pointer input.
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of this service

### **Allowed From**

Threads

### **Example**

```
/* Stop the TFTP Server called "my_server". */
status = nxd_tftp_server_stop(&my_server);

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

### **See Also**

nxd\_tftp\_client\_create, nxd\_tftp\_client\_delete,  
 nxd\_tftp\_client\_error\_info\_get, nxd\_tftp\_client\_file\_close,  
 nxd\_tftp\_client\_file\_open, nxd\_tftp\_client\_file\_read,  
 nxd\_tftp\_client\_file\_write, nxd\_tftp\_client\_packet\_allocate,  
 nxd\_tftp\_server\_create, nxd\_tftp\_server\_delete, nxd\_tftp\_server\_start