Trivial File Transfer Protocol (TFTP)

User Guide

Express Logic, Inc.

858.613.6640
Toll Free 888.THREADX
FAX 858.521.4259

www.expresslogic.com

Part Number: 000-1052
Revision 5.3

Contents

Chapter 1

Introduction to 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.

TFTP Requirements

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

The TFTP Server portion of the NetX TFTP package has several additional requirements. First, it requires complete access to 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 (usually FileX). 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 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:

ooooof…f0OCTET0

Where:

oooo 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:

oooobbbbd…d

Where:

oooo 2-byte Opcode field

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

bbbb 2-byte Block Number field (1-n)

d…d 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

The TFTP Server utilizes the well-known UDP port 69 to field client requests. TFTP Clients may use any available UDP port. Data packets are fixed at 512 bytes, until the last packet. 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. Client Issues “Open For Read” request with the File Name and waits for a packet from Server.
2. Server sends the first 512 bytes of the file.
3. Client receives data, sends ACK, and waits for the next packet if the last packet had 512 bytes.
4. The sequence ends when a packet containing fewer than 512 bytes is received.

TFTP Write Requests:

1. Client Issues “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. Client sends the first 512 bytes of the file to the Server and waits for an ACK.
4. Server sends 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 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 TFTP is compliant with RFC1350 and related RFCs.

Chapter 2

Installation and Use of TFTP

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

Product Distribution

TFTP 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:

nx_tftp.h	Header file for TFTP for NetX
nx_tftp_client.c	C Source file for TFTP Client for NetX
nx_tftp_server.c	C Source file for TFTP Server for NetX
filex_stub.h	Stub file if FileX is not present
nx_tftp.pdf	PDF description of TFTP for NetX
demo_netx_tftp.c	NetX TFTP demonstration

TFTP Installation

In order to use TFTP 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_tftp.h*, *nx_tftp_client.c*, and *nx_tftp_server.c* files should be copied into this directory.

Using TFTP

Using TFTP for NetX is easy. Basically, the application code must include *nx_tftp.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_tftp.h* is included, the application code is then able to make the TFTP function calls specified later in this guide. The application must also include *nx_tftp_client.c* and *nx_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 TFTP.

Note that since TFTP utilizes NetX 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 TFTP is described in Figure 1.1 that appears below. In this example, the TFTP include file *nx_tftp.h* is 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. After successful creation, a 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.

```
0001 #include   "tx_api.h"
0002 #include   "nx_api.h"
0003 #include   "fx_api.h"
0004 #include   "nx_tftp.h"
0005
0006 #define     DEMO_STACK_SIZE         2048
0007
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
0027 ULONG                   error_counter;
0028
0029
0030 /* Define the memory area for the FileX RAM disk.  */
0031
0032 UCHAR                   ram_disk_memory[32000];
0033
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                 17, 17, 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), 0xFFFFFF00UL,
0096                                   &server_pool, _nx_ram_network_driver,
0097                                   pointer, 2048, 1);
0098     pointer = pointer + 2048;
0099
0100     /* Check for errors.  */
0101     if (status)
0102         error_counter++;
0103
0104     /* Enable ARP and supply ARP cache memory for IP Instance 0.  */
0105     status =  nx_arp_enable(&server_ip, (void *) pointer, 1024);
0106     pointer = pointer + 1024;
0107
0108     /* Check for errors.  */
0109     if (status)
0110         error_counter++;
0111
0112     /* Enable UDP.  */
0113     status =  nx_udp_enable(&server_ip);
0114
0115     /* Check for errors.  */
0116     if (status)
0117         error_counter++;
0118
0119     /* Create the TFTP server.  */
0120     status =  nx_tftp_server_create(&server, "TFTP Server Instance",
0121           &server_ip, &ram_disk, pointer, DEMO_STACK_SIZE, &server_pool);
0122     pointer =  pointer + DEMO_STACK_SIZE;
0123
0124     /* Check for errors.  */
0125     if (status)
0126         error_counter++;
0127
0128     /* Start the TFTP server.  */
0129     status =  nx_tftp_server_start(&server);
0130
0131     /* Check for errors.  */
0132     if (status)
0133         error_counter++;
0134
0135     /* Create a packet pool for the TFTP client.  */
0136     status =  nx_packet_pool_create(&client_pool, "NetX Client Packet Pool",
0137                                                 560, pointer, 8192);
0138     pointer =  pointer + 8192;
0139
0140     /* Create an IP instance for the TFTP client.  */
0141     status = nx_ip_create(&client_ip, "NetX Client IP Instance",
0142                                       IP_ADDRESS(1, 2, 3, 5), 0xFFFFFF00UL,
```

```
0143                     &client_pool, _nx_ram_network_driver, pointer, 2048, 1);
0144     pointer = pointer + 2048;
0145
0146     /* Enable ARP and supply ARP cache memory for IP Instance 1.  */
0147     status =  nx_arp_enable(&client_ip, (void *) pointer, 1024);
0148     pointer = pointer + 1024;
0149
0150     /* Enable UDP for client IP instance.  */
0151     status =  nx_udp_enable(&client_ip);
0152 }
0153
0154
0155 /* Define the TFTP client thread.  */
0156
0157 void    client_thread_entry(ULONG thread_input)
0158 {
0159
0160 NX_PACKET   *my_packet;
0161 UINT        status;
0162
0163
0164     /* Create a TFTP client.  */
0165     status =  nx_tftp_client_create(&client, "New Client", &client_ip,
0166                                                             &client_pool);
0167
0168     /* Check status.  */
0169     if (status)
0170         error_counter++;
0171
0172     /* Open a TFTP file for writing.  */
0173     status =  nx_tftp_client_file_open(&client, "test.txt",
0174                 IP_ADDRESS(1,2,3,4), NX_TFTP_OPEN_FOR_WRITE, 100);
0175
0176     /* Check status.  */
0177     if (status)
0178         error_counter++;
0179
0180     /* Allocate a TFTP packet.  */
0181     status =  nx_tftp_client_packet_allocate(&client_pool, &my_packet, 100);
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                                         "ABCDEFGHIJKLMNOPQRSTUVWXYZ  ", 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 =  nx_tftp_client_file_write(&client, my_packet, 100);
0197
0198     /* Check status.  */
0199     if (status)
0200         error_counter++;
0201
0202     /* Close this file.  */
0203     status =  nx_tftp_client_file_close(&client);
0204
0205     /* Check status.  */
0206     if (status)
0207         error_counter++;
0208
0209     /* Open the same file for reading.  */
0210     status =  nx_tftp_client_file_open(&client, "test.txt",
0211                         IP_ADDRESS(1,2,3,4), NX_TFTP_OPEN_FOR_READ, 100);
0212
0213     /* Check status.  */
0214     if (status)
0215         error_counter++;
0216
0217     /* Read the file back.  */
0218     status =  nx_tftp_client_file_read(&client, &my_packet, 100);
0219
0220     /* Check status.  */
0221     if (status)
0222         error_counter++;
0223     else
```

```
0224        nx_packet_release(my_packet);
0225
0226    /* Close the file again.  */
0227    status =  nx_tftp_client_file_close(&client);
0228
0229    /* Check status.  */
0230    if (status)
0231        error_counter++;
0232
0233    /* Delete the client.  */
0234    status =  nx_tftp_client_delete(&client);
0235
0236    /* Check status.  */
0237    if (status)
0238        error_counter++;
0239 }
```

Figure 1.1 Example of TFTP use with NetX

Configuration Options

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

Define	Meaning
NX_DISABLE_ERROR_CHECKING	Defined, this option removes the basic TFTP error checking. It is typically used after the application has been debugged.
NX_TFTP_SERVER_PRIORITY	The priority of the TFTP server thread. By default, this value is defined as 16 to specify priority 16.
NX_TFTP_MAX_CLIENTS	The maximum number of clients the server can handle at one time. By default, this value is 10 to support 10 clients at once.
NX_TFTP_ERROR_STRING_MAX	The maximum number of characters in the error string. By default, this value is 64.
NX_TFTP_NO_FILEX	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.
NX_TFTP_TYPE_OF_SERVICE	Type of service required for the TFTP UDP 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_tftp.h*.

NX_TFTP_FRAGMENT_OPTION	Fragment enable for TFTP UDP requests. By default, this value is NX_DONT_FRAGMENT to disable TFTP UDP fragmenting. This define can be set by the application prior to inclusion of *nx_tftp.h*.
NX_TFTP_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_tftp.h.*

Chapter 3

Description of TFTP Services

This chapter contains a description of all NetX TFTP 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_tftp_client_create
Create a TFTP client instance

nx_tftp_client_delete
Delete a TFTP client instance

nx_tftp_client_error_info_get
Get client error information

nx_tftp_client_file_close
Close client file

nx_tftp_client_file_open
Open client file

nx_tftp_client_file_read
Read a block from client file

nx_tftp_client_file_write
Write block to client file

nx_tftp_client_packet_allocate
Allocate packet for client file write

nx_tftp_server_create
Create TFTP server

nx_tftp_server_delete
Delete TFTP server

nx_tftp_server_start

Start TFTP Server

nx_tftp_server_stop
Stop TFTP Server

nx_tftp_client_create

Create a TFTP client instance

Prototype

```
UINT nx_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

tftp_client_ptr	Pointer to TFTP client control block.
tftp_client_name	Name of this TFTP client instance
ip_ptr	Pointer to previously created IP instance.
pool_ptr	Pointer to packet pool TFTP client instance.

Return Values

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

Allowed From

Initialization and Threads

Example

```
/* Create a TFTP client instance.  */
status =  nx_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

nx_tftp_client_delete, nx_tftp_client_error_info_get, nx_tftp_client_file_close, nx_tftp_client_file_open, nx_tftp_client_file_read, nx_tftp_client_file_write, nx_tftp_client_packet_allocate, nx_tftp_server_create, nx_tftp_server_delete, nx_tftp_server_start, nx_tftp_server_stop

nx_tftp_client_delete

Delete a TFTP client instance

Prototype

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

Description

This service deletes a previously created TFTP client instance.

Input Parameters

tftp_client_ptr	Pointer to previously created TFTP client instance.

Return Values

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

Allowed From

Threads

Example

```
/* Delete a TFTP client instance.  */
status =  nx_tftp_client_delete(&my_tftp_client);

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

See Also

nx_tftp_client_create, nx_tftp_client_error_info_get, nx_tftp_client_file_close, nx_tftp_client_file_open, nx_tftp_client_file_read, nx_tftp_client_file_write, nx_tftp_client_packet_allocate, nx_tftp_server_create, nx_tftp_server_delete, nx_tftp_server_start, nx_tftp_server_stop

nx_tftp_client_error_info_get

Get client error information

Prototype

```
UINT nx_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

tftp_client_ptr	Pointer to previously created TFTP client instance.
error_code	Pointer to destination area for error code
error_string	Pointer to destination for error string

Return Values

NX_SUCCESS	(0x00)	Successful TFTP error info get.
NX_PTR_ERROR	(0x16)	Invalid TFTP client pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Get error information for client.  */
status =  nx_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

nx_tftp_client_create, nx_tftp_client_delete, nx_tftp_client_file_close, nx_tftp_client_file_open, nx_tftp_client_file_read, nx_tftp_client_file_write, nx_tftp_client_packet_allocate, nx_tftp_server_create, nx_tftp_server_delete, nx_tftp_server_start, nx_tftp_server_stop

nx_tftp_client_file_close

Close client file

Prototype

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

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

tftp_client_ptr	Pointer to previously created TFTP client instance.

Return Values

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

Allowed From

Threads

Example

```
/* Close the previously opened file associated with “my_client”.  */
status =  nx_tftp_client_file_close(&my_tftp_client);

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

See Also

nx_tftp_client_create, nx_tftp_client_delete, nx_tftp_client_error_info_get, nx_tftp_client_file_open, nx_tftp_client_file_read, nx_tftp_client_file_write, nx_tftp_client_packet_allocate, nx_tftp_server_create, nx_tftp_server_delete, nx_tftp_server_start, nx_tftp_server_stop

nx_tftp_client_file_open

Open client file

Prototype

```
UINT nx_tftp_client_file_open(NX_TFTP_CLIENT *tftp_client_ptr,
            CHAR *file_name, ULONG 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.

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 IP address of TFTP Server.

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 0xFFFFFFFE)
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 (1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the TFTP server response.

Return Values

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

Allowed From

Threads

Example

```
/* Open file “test.txt” for reading on the TFTP server at 202.2.2.13.  */
status =  nx_tftp_client_file_open(&my_tftp_client, “test.txt”,
               IP_ADDRESS(202,2,2,13), NX_TFTP_OPEN_FOR_READ, 200);

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

See Also

nx_tftp_client_create, nx_tftp_client_delete, nx_tftp_client_error_info_get, nx_tftp_client_file_close, nx_tftp_client_file_read, nx_tftp_client_file_write, nx_tftp_client_packet_allocate, nx_tftp_server_create, nx_tftp_server_delete, nx_tftp_server_start, nx_tftp_server_stop

nx_tftp_client_file_read

Read a block from client file

Prototype

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

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

tftp_client_ptr	Pointer to TFTP client control block.
packet_ptr	Destination for packet containing the block read from the file.
wait_option	Defines how long the service will wait for the read to complete. The wait options are defined as follows: **timeout value** (0x00000001 through 0xFFFFFFFE) **TX_WAIT_FOREVER** (0xFFFFFFFF) Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the TFTP server responds to the request. Selecting a numeric value (1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the TFTP server to send a block of the file.

Return Values

NX_SUCCESS	(0x00)	Successful TFTP client block read
NX_TFTP_NOT_OPEN	(0xC3)	Specified TFTP client file is not open for reading

NX_NO_PACKET	(0x01)	Packet not received fromTFTP server.
NX_INVALID_TFTP_SERVER_ADDRESS	**(0x08)**	Invalid server address received
NX_TFTP_NO_ACK_RECEIVED	**(0x09)**	No ACK received from server
status		Actual NetX completion status
NX_TFTP_END_OF_FILE	(0xC5)	End of file detected.
NX_PTR_ERROR	(0x16)	Invalid TFTP client pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

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

/* If status is NX_SUCCESS a block of the TFPT file is in the payload of
   “return_packet_ptr”.  */
```

See Also

nx_tftp_client_create, nx_tftp_client_delete, nx_tftp_client_error_info_get, nx_tftp_client_file_close, nx_tftp_client_file_open, nx_tftp_client_file_write, nx_tftp_client_packet_allocate, nx_tftp_server_create, nx_tftp_server_delete, nx_tftp_server_start, nx_tftp_server_stop

nx_tftp_client_file_write

Write block to client file

Prototype

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

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

tftp_client_ptr	Pointer to TFTP client control block.
packet_ptr	Packet containing the block to write to the file.
wait_option	Defines how long the service will wait for the write to complete. The wait options are defined as follows: **timeout value** (0x00000001 through 0xFFFFFFFE) **TX_WAIT_FOREVER** (0xFFFFFFFF) Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the TFTP server responds to the request. Selecting a numeric value (1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the TFTP server to send an ACK for the write request.

Return Values

NX_SUCCESS	(0x00)	Successful Client block write
NX_TFTP_NOT_OPEN	(0xC3)	Specified TFTP client file is not open for writing

NX_TFTP_TIMEOUT	(0xC1)	Timeout waiting for Server ACK
NX_INVALID_TFTP_SERVER_ADDRESS	**(0x08)**	Invalid server address received
NX_TFTP_NO_ACK_RECEIVED	**(0x09)**	No ACK received from server
status		Actual NetX completion status
NX_PTR_ERROR	(0x16)	Invalid TFTP client pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Write a block to the previously opened file of “my_client”.  */
status =  nx_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 TFPT file opened by “my_client”.  */
```

See Also

nx_tftp_client_create, nx_tftp_client_delete, nx_tftp_client_error_info_get, nx_tftp_client_file_close, nx_tftp_client_file_open, nx_tftp_client_file_read, nx_tftp_client_packet_allocate, nx_tftp_server_create, nx_tftp_server_delete, nx_tftp_server_start, nx_tftp_server_stop

nx_tftp_client_packet_allocate

Allocate packet for client file write

Prototype

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

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

pool_ptr Pointer to packet pool.

packet_ptr Destination for pointer to allocated packet.

wait_option Defines how long the service will wait for the packet allocate to complete. The wait options are defined as follows:

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

Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the allocation completes.

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

Return Values

NX_SUCCESS	(0x00)	Successful packet allocate
NX_NO_PACKET	(0x01)	No packet available
NX_WAIT_ABORTED	(0x1A)	Requested suspension was aborted by a call to *tx_thread_wait_abort*.

status		Actual NetX completion status
NX_PTR_ERROR	(0x16)	Invalid TFTP client pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Allocate a packet for TFTP file write.  */
status =  nx_tftp_packet_allocate(&my_pool, &packet_ptr, 200);

/* If status is NX_SUCCESS “packet_ptr” contains the new packet.  */
```

See Also

nx_tftp_client_create, nx_tftp_client_delete, nx_tftp_client_error_info_get, nx_tftp_client_file_close, nx_tftp_client_file_open, nx_tftp_client_file_read, nx_tftp_client_file_write, nx_tftp_server_create, nx_tftp_server_delete, nx_tftp_server_start, nx_tftp_server_stop

nx_tftp_server_create

Create TFTP server

Prototype

```
UINT nx_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 *nx_tftp_server_start*.

Important Note: The application must make certain the supplied IP, 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

tftp_server_ptr	Pointer to TFTP server control block.
tftp_server_name	Name of this TFTP server instance
ip_ptr	Pointer to previously created IP instance.
media_ptr	Pointer to FileX media instance.
stack_ptr	Pointer to stack area for TFTP server thread.
stack_size	Number of bytes in the TFTP server stack.
pool_ptr	Pointer to TFTP packet pool. Note that the supplied pool must have packet payloads at least 560 bytes in size.[1]

Return Values

NX_SUCCESS	(0x00)	Successful TFTP server create

[1] The data portion of a packet is exactly 512 bytes, but the packet payload size must be at least 560 bytes. The remaining bytes are used for the UDP, IP, and Ethernet headers.

NX_TFTP_POOL_ERROR	(0xC6)	Packet pool has packet size of less than 560 bytes
status		Actual NetX completion status
NX_PTR_ERROR	(0x16)	Invalid TFTP server, stack, IP, media, or pool pointer.

Allowed From

Initialization, Threads

Example

```
/* Create a TFTP server called “my_server”.  */
status =  nx_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

nx_tftp_client_create, nx_tftp_client_delete, nx_tftp_client_error_info_get, nx_tftp_client_file_close, nx_tftp_client_file_open, nx_tftp_client_file_read, nx_tftp_client_file_write, nx_tftp_client_packet_allocate, nx_tftp_server_delete, nx_tftp_server_start, nx_tftp_server_stop

nx_tftp_server_delete

Delete TFTP server

Prototype

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

Description

This service deletes a previously created TFTP server.

Input Parameters

tftp_server_ptr	Pointer to TFTP server control block.

Return Values

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

Allowed From

Threads

Example

```
/* Delete the TFTP server called “my_server”.  */
status =  nx_tftp_server_delete(&my_server);

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

See Also

nx_tftp_client_create, nx_tftp_client_delete, nx_tftp_client_error_info_get, nx_tftp_client_file_close, nx_tftp_client_file_open, nx_tftp_client_file_read, nx_tftp_client_file_write, nx_tftp_client_packet_allocate, nx_tftp_server_create, nx_tftp_server_start, nx_tftp_server_stop

nx_tftp_server_start

Start TFTP server

Prototype

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

Description

This service starts the previously created TFTP server.

Input Parameters

tftp_server_ptr	Pointer to TFTP server control block.

Return Values

NX_SUCCESS	(0x00)	Successful TFTP server start
NX_PTR_ERROR	(0x16)	Invalid TFTP server pointer.

Allowed From

Initialization, threads

Example

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

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

See Also

nx_tftp_client_create, nx_tftp_client_delete, nx_tftp_client_error_info_get, nx_tftp_client_file_close, nx_tftp_client_file_open, nx_tftp_client_file_read, nx_tftp_client_file_write, nx_tftp_client_packet_allocate, nx_tftp_server_create, nx_tftp_server_delete, nx_tftp_server_stop

nx_tftp_server_stop

Stop TFTP server

Prototype

```
UINT nx_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

NX_SUCCESS	(0x00)	Successful TFTP server stop
NX_PTR_ERROR	(0x16)	Invalid TFTP server pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

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

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

See Also

nx_tftp_client_create, nx_tftp_client_delete, nx_tftp_client_error_info_get, nx_tftp_client_file_close, nx_tftp_client_file_open, nx_tftp_client_file_read, nx_tftp_client_file_write, nx_tftp_client_packet_allocate, nx_tftp_server_create, nx_tftp_server_delete, nx_tftp_server_start