NETX Duo

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

Part Number: 000-1052
Revision 5.2

Contents

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:

nxd_tftp_client.h	Header file for NetX Duo TFTP Client
nxd_tftp_client.c	C Source file for NetX Duo TFTP Client
nxd_tftp_server.h	Header file for NetX Duo TFTP Server
nxd_tftp_server.c	C Source file for NetX Duo TFTP Server
filex_stub.h	Stub file if FileX is not present
nxd_tftp.pdf	PDF description of NetX Duo TFTP
demo_netxduo_tftp.c	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. Tthe 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
```

```

/* Define the counters used in the demo application...  */
ULONG                   error_counter;

/* Define the memory area for the FileX RAM disk.  */
UCHAR                   ram_disk_memory[32000];

/* Define which IP protocol to use for TFTP, IPv4 or IPv6.  To use
  IPv6,NetX Duo must be enabled with IPv6. */
#define IP_TYPE          4

/* Define function prototypes.  */

VOID    _fx_ram_driver(FX_MEDIA *media_ptr);
VOID    _nx_ram_network_driver(NX_IP_DRIVER *driver_req_ptr);
void    client_thread_entry(ULONG thread_input);


/* Define main entry point.  */

int main()
{

    /* Enter the ThreadX kernel.  */
    tx_kernel_enter();
}


/* Define what the initial system looks like.  */

void    tx_application_define(void *first_unused_memory)
{

UINT    status;
UCHAR   *pointer;


    /* Setup the working pointer.  */
    pointer =  (UCHAR *) first_unused_memory;

    /* Create the main TFTP demo thread.  */
    status = tx_thread_create(&client_thread, "thread 0",
                 client_thread_entry, 0, pointer, DEMO_STACK_SIZE,
                 6, 6, TX_NO_TIME_SLICE, TX_AUTO_START);
    pointer += DEMO_STACK_SIZE ;

    /* Check for errors.  */
    if (status)
        error_counter++;

    /* Open the RAM disk.  */
    status = fx_media_open(&ram_disk, "RAM DISK", _fx_ram_driver,
                                ram_disk_memory, pointer, 4096);
    pointer += 4096 ;

    /* Check for errors.  */
    if (status)
       error_counter++;

    /* Create the packet pool for the TFTP Server.  The packet size must
       be a minimum of 560 bytes.  */
    status = nx_packet_pool_create(&server_pool, "NetX Server Packet Pool",
                                                 560, pointer, 8192);
    pointer = pointer + 8192;

    /* Check for errors.  */
    if (status)
        error_counter++;

    /* Create the IP instance for the TFTP Server.  */
    status = nx_ip_create(&server_ip, "NetX Server IP Instance",
                                   IP_ADDRESS(1,2,3,4), 0xFFFFFF00UL,
                                   &server_pool, _nx_ram_network_driver,
                                   pointer, 2048, 1);
    pointer = pointer + 2048;

    /* Check for errors.  */
    if (status)
        error_counter++;

    /* Enable ARP and supply ARP cache memory for IP Instance 0.  */
    status =  nx_arp_enable(&server_ip, (void *) pointer, 1024);
    pointer = pointer + 1024;

    /* Check for errors.  */
    if (status)
        error_counter++;

    /* Enable UDP.  */
    status =  nx_udp_enable(&server_ip);

    /* Check for errors.  */
    if (status)
        error_counter++;

    /* Create the TFTP server.  */
    status =  nxd_tftp_server_create(&server, "TFTP Server Instance",
          &server_ip, &ram_disk, pointer, DEMO_STACK_SIZE, &server_pool);
    pointer =  pointer + DEMO_STACK_SIZE;

    /* Check for errors.  */
    if (status)
        error_counter++;

    /* Start the TFTP server.  */
    status =  nxd_tftp_server_start(&server);

    /* Check for errors.  */
    if (status)
        error_counter++;

    /* Create a packet pool for the TFTP client.  */
    status =  nx_packet_pool_create(&client_pool, "NetX Client Packet Pool",
                                              560, pointer, 8192);
    pointer =  pointer + 8192;

    /* Create an IP instance for the TFTP client.  */
    status = nx_ip_create(&client_ip, "NetX Client IP Instance",
                                        IP_ADDRESS(1, 2, 3, 5), 0xFFFFFF00UL,
                    &client_pool, _nx_ram_network_driver, pointer, 2048, 1);
    pointer = pointer + 2048;

    /* Enable ARP and supply ARP cache memory for IP Instance 1.  */
    status =  nx_arp_enable(&client_ip, (void *) pointer, 1024);
    pointer = pointer + 1024;

    /* Enable UDP for client IP instance.  */
    status =  nx_udp_enable(&client_ip);
}


/* Define the TFTP Client thread.  */

void    client_thread_entry(ULONG thread_input)
{

NX_PACKET   *my_packet;
UINT        status;


    /* Create a TFTP client.  */
    status =  nxd_tftp_client_create(&client, "New Client", &client_ip,
                                                         &client_pool);

    /* Check status.  */
    if (status)
        error_counter++;

    /* Open a TFTP file for writing.  */
    status =  nxd_tftp_client_file_open(&client, "test.txt",
                 IP_ADDRESS(1,2,3,4), NX_TFTP_OPEN_FOR_WRITE, 300, IP_TYPE);

    /* Check status.  */
    if (status)
        error_counter++;

    /* Allocate a TFTP packet.  */
    status =  nxd_tftp_client_packet_allocate(&client_pool, &my_packet, 100,
                                               IP_TYPE);

    /* Check status.  */
    if (status)
```

```
        error_counter++;

    /* Write ABCs into the packet payload!  */
    memcpy(my_packet -> nx_packet_prepend_ptr,
                                    "ABCDEFGHIJKLMNOPQRSTUVWXYZ  ", 28);

    /* Adjust the write pointer.  */
    my_packet -> nx_packet_length =  28;
    my_packet ->nx_packet_append_ptr = my_packet -> nx_packet_prepend_ptr+28;

    /* Write this packet to the file via TFTP.  */
    status =  nxd_tftp_client_file_write(&client, my_packet, 300, IP_TYPE);

    /* Check status.  */
    if (status)
        error_counter++;

    /* Close this file.  */
    status =  nxd_tftp_client_file_close(&client, IP_TYPE);

    /* Check status.  */
    if (status)
        error_counter++;

    /* Open the same file for reading.  */
    status =  nxd_tftp_client_file_open(&client, "test.txt",
                      IP_ADDRESS(1,2,3,4), NX_TFTP_OPEN_FOR_READ, 200,
                     IP_TYPE);

    /* Check status.  */
    if (status)
        error_counter++;

    /* Read the file back.  */
    status =  nxd_tftp_client_file_read(&client, &my_packet, 200, IP_TYPE);

    /* Check status.  */
    if (status)
        error_counter++;
    else
       nx_packet_release(my_packet);

    /* Close the file again.  */
    status =  nxd_tftp_client_file_close(&client) , IP_TYPE);

    /* Check status.  */
    if (status)
        error_counter++;

    /* Delete the client.  */
    status =  nxd_tftp_client_delete(&client);

    /* Check status.  */
    if (status)
        error_counter++;
}
```

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
NX_DISABLE_ERROR_CHECKING	This option removes the basic TFTP error checking. It is typically used after the application has been debugged. By default it is not defined.
NX_TFTP_SERVER_PRIORITY	The priority of the TFTP server thread. The default value is 16 to specify priority 16 in *nxd_tftp_server.h*.
NX_TFTP_MAX_CLIENTS	The maximum number of clients the server can handle at one time. The default value is 10 in *nxd_tftp_server.h*.
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 requests. By default, this value is defined as NX_IP_NORMAL to indicate normal IP packet service.
NX_TFTP_FRAGMENT_OPTION	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

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 =  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

tftp_client_ptr	Pointer to previously created TFTP client instance.

Return Values

NX_SUCCESS	(0x00)	Successful TFTP Client delete.
NX_PTR_ERROR	(0x16)	Invalid pointer input.
NX_CALLER_ERROR	(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

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 =  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

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

Return Values

NX_SUCCESS	(0x00)	Successful TFTP file close.
status		Actual NetX completion status
NX_PTR_ERROR	(0x16)	Invalid pointer input.
NX_CALLER_ERROR	(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 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.

ip_type	Indicate which IP protocol to use. Valid options are IPv4 (4) or IPv6 (6).

Return Values

NX_SUCCESS	(0x00)	Successful 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
NX_PTR_ERROR	(0x16)	Invalid pointer input.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service
NX_IP_ADDRESS_ERROR	(0x21)	Invalid Server IP address
NX_OPTION_ERROR	(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 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.
ip_type	Indicate which IP protocol to use. Valid options

are IPv4 (4) or IPv6 (6).

Return Values

NX_SUCCESS	(0x00)	Successful 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 completion status
NX_PTR_ERROR	(0x16)	Invalid pointer input.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service
NX_IP_ADDRESS_ERROR	(0x21)	Invalid Server IP address
NX_OPTION_ERROR	(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

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.
ip_type	Indicate which IP protocol to use. Valid options are IPv4 (4) or IPv6 (6).

Return Values

NX_SUCCESS	(0x00)	Successful Client block read

NX_TFTP_NOT_OPEN	(0xC3)	Specified Client file is not open for reading

NX_NO_PACKET	(0x01)	No Packet received from Server.
NX_INVALID_TFTP_SERVER_ADDRESS	**(0x08)**	Invalid server address received
NX_TFTP_NO_ACK_RECEIVED	**(0x09)**	No ACK received from Server
status		Actual completion status
NX_TFTP_END_OF_FILE	(0xC5)	End of file detected.
NX_PTR_ERROR	(0x16)	Invalid pointer input.
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 =  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

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.
ip_type	Indicate which IP protocol to use. Valid options are IPv4 (4) or IPv6 (6).

Return Values

NX_SUCCESS	(0x00)	Successful Client block write
NX_TFTP_NOT_OPEN	(0xC3)	Specified 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 completion status
NX_PTR_ERROR	(0x16)	Invalid pointer input.
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 =  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

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.
ip_type	Indicate which IP protocol to use. Valid options are IPv4 (4) or IPv6 (6).

Return Values

NX_SUCCESS	(0x00)	Successful packet allocate

NX_NO_PACKET	(0x01)	No packet available
status		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

tftp_client_ptr	Pointer to TFTP Client instance
if_index	Index of physical interface to use

Return Values

NX_SUCCESS	(0x00)	Successfully set interface
NX_TFTP_INVALID_INTERFACE	(0x0B)	Invalid interface input
NX_PTR_ERROR	(0x16)	Invalid pointer input.
NX_CALLER_ERROR	(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

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 TFTP Server stack area.
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 580 bytes in size.[1]

Return Values

NX_SUCCESS	(0x00)	Successful Server create

[1] 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.

NX_TFTP_POOL_ERROR	(0xC6)	Packet pool has packet size of less than 560 bytes
status		Actual completion status
NX_PTR_ERROR	(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

tftp_server_ptr	Pointer to TFTP Server control block.

Return Values

NX_SUCCESS	(0x00)	Successful Server delete
NX_PTR_ERROR	(0x16)	Invalid pointer input.
NX_CALLER_ERROR	(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

tftp_server_ptr	Pointer to TFTP Server control block.

Return Values

NX_SUCCESS	(0x00)	Successful Server start
NX_PTR_ERROR	(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

NX_SUCCESS	(0x00)	Successful Server stop
NX_PTR_ERROR	(0x16)	Invalid pointer input.
NX_CALLER_ERROR	(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