

NetX Duo™

Trivial File Transfer Protocol (TFTP) for NetX Duo

User Guide

Renesas Synergy™ Platform

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (<http://www.renesas.com>).

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
"Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.

6. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

Renesas Synergy Specific Information

If you are using NetX Duo TFTP for the Renesas Synergy platform, please use the following information.

Multi-Thread Support

Page 7: Multi-thread support for the TFTP client has not been tested with SSP v1.5.0..

Product Distribution

Page 8: The distribution of TFTP included with the Renesas Synergy SSP installation does not include the file **demo_netxdue_tftp.c**. Please ignore references to this file.

NetX Duo TFTP Installation

Page 8: If you are using Renesas Synergy SSP and the e² studio ISDE, TFTP will already be installed. You can ignore the Installation and Use of NetX Duo TFTP section.



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

©2002-2018 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 5.11

Contents

Chapter 1 Introduction to NetX Duo TFTP	4
TFTP Requirements	4
TFTP File Names	4
TFTP Messages	5
TFTP Communication	6
TFTP Server Session Timer	6
TFTP Multi-Thread Support	7
TFTP RFCs	7
Chapter 2 Installation and Use of NetX Duo TFTP	8
Product Distribution	8
TFTP Installation	8
Using TFTP	8
Small Example System	9
Configuration Options	17
Chapter 3 Description of TFTP Services	20
nxd_tftp_client_create	22
nxd_tftp_client_delete	24
nxd_tftp_client_error_info_get	25
nxd_tftp_client_file_close	26
nx_tftp_client_file_open	27
nxd_tftp_client_file_open	29
nxd_tftp_client_file_read	31
nxd_tftp_client_file_write	33
nxd_tftp_client_packet_allocate	35
nxd_tftp_client_set_interface	37
nxd_tftp_server_create	38
nxd_tftp_server_delete	40
nxd_tftp_server_start	41
nxd_tftp_server_stop	42

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.

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:

oooof...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

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.
- 6.

TFTP Server Session Timer

The TFTP Server has a limited number of client request slots. If a client session appears to be dropped, that slot cannot be available for re-use. However if the `NX_TFTP_SERVER_RETRANSMIT_ENABLE` option is enabled, the NetX Duo TFTP Server creates a session timer that monitors the timeout on each of its client sessions. When a session

timeout expires it is terminated and any open files are closed. Thus the 'slot' becomes available for another TFTP Client request.

To set the timeout, adjust the configuration option `NX_TFTP_SERVER_RETRANSMIT_TIMEOUT` which by default is 200 timer ticks. The interval between which session timeouts are checked is set by the `NX_TFTP_SERVER_TIMEOUT_PERIOD` which is 20 timer ticks by default.

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:

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

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 TFTP

To run a TFTP application, 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 19 and 20. Next, the TFTP Server is created in “*tx_application_define*” at line 179. Note that the TFTP Server control block “*server*” was defined as a global variable at line 45 previously. This demo chooses to use IPv4 for its TFTP communication in line 14. After successful creation, the TFTP Server is started at line 303. At line 397 the TFTP Client is created. And finally, the Client writes the file at line 436 and reads the file back at line 471.

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
```

```
1 /* This is a small demo of TFTP on the high-performance NetX TCP/IP stack.
This demo
2   relies on ThreadX and NetX , to show a simple file transfer from the client
3   and then back to the server. */
4
5 /* Indicate if using a NetX TFTP services. To port a NetX TFTP application to
NetX TFTP
6   undefine this term. */
7
8
9 #define USE_DUO
10
11 /* If the host application is using NetX Duo, determine which IP version to
use.
12   Make sure IPv6 in NetX Duo is enabled if planning to use TFTP over IPv6 */
13 #ifdef USE_DUO
14 #define IP_TYPE 6
15 #endif /* USE_DUO */
16
17 #include "tx_api.h"
18 #include "nx_api.h"
19 #include "nxd_tftp_client.h"
```

```

20 #include      "nxd_tftp_server.h"
21 #ifndef      NX_TFTP_NO_FILEX
22 #include      "fx_api.h"
23 #endif
24
25
26 #define      DEMO_STACK_SIZE      4096
27
28 /* To use another file storage utility define this symbol:
29 #define NX_TFTP_NO_FILEX
30 */
31
32 /* Define the ThreadX, NetX, and FileX object control blocks... */
33
34 TX_THREAD      server_thread;
35 TX_THREAD      client_thread;
36 NX_PACKET_POOL server_pool;
37 NX_IP           server_ip;
38 NX_PACKET_POOL client_pool;
39 NX_IP           client_ip;
40 FX_MEDIA        ram_disk;
41
42 /* Define the NetX TFTP object control blocks. */
43
44 NX_TFTP_CLIENT  client;
45 NX_TFTP_SERVER  server;
46
47 /* Define the application global variables */
48
49 #define          CLIENT_ADDRESS  IP_ADDRESS(1, 2, 3, 5)
50 #define          SERVER_ADDRESS  IP_ADDRESS(1, 2, 3, 4)
51
52 NXD_ADDRESS     server_ip_address;
53 NXD_ADDRESS     client_ip_address;
54
55 UINT            error_counter = 0;
56
57 /* Define buffer used in the demo application. */
58 UCHAR           buffer[255];
59 ULONG           data_length;
60
61
62 /* Define the memory area for the FileX RAM disk. */
63 #ifndef NX_TFTP_NO_FILEX
64 UCHAR           ram_disk_memory[32000];
65 UCHAR           ram_disk_sector_cache[512];
66 #endif
67
68
69 /* Define function prototypes. */
70
71 VOID            _fx_ram_driver(FX_MEDIA *media_ptr);
72 VOID            _nx_ram_network_driver(NX_IP_DRIVER *driver_req_ptr);
73 void            client_thread_entry(ULONG thread_input);
74 void            server_thread_entry(ULONG thread_input);
75
76
77 /* Define main entry point. */
78
79 int main()
80 {
81     /* Enter the ThreadX kernel. */
82     tx_kernel_enter();
83 }
84
85
86
87 /* Define what the initial system looks like. */
88
89 void            tx_application_define(void *first_unused_memory)
90 {
91     UINT         status;
92     UCHAR        *pointer;
93
94     /* Setup the working pointer. */
95     pointer = (UCHAR *) first_unused_memory;
96
97     /* Create the main TFTP server thread. */

```

```

101     status = tx_thread_create(&server_thread, "TFTP Server Thread",
server_thread_entry, 0,
102                               pointer, DEMO_STACK_SIZE,
103                               4, 4, TX_NO_TIME_SLICE, TX_AUTO_START);
104
105     pointer += DEMO_STACK_SIZE ;
106
107     /* Check for errors. */
108     if (status)
109         error_counter++;
110
111
112     /* Create the main TFTP client thread at a slightly lower priority. */
113     status = tx_thread_create(&client_thread, "TFTP Client Thread",
client_thread_entry, 0,
114                               pointer, DEMO_STACK_SIZE,
115                               5, 5, TX_NO_TIME_SLICE, TX_DONT_START);
116
117     pointer += DEMO_STACK_SIZE ;
118
119     /* Check for errors. */
120     if (status)
121         error_counter++;
122
123     /* Initialize the NetX system. */
124     nx_system_initialize();
125
126     /* Note: The data portion of a packet is exactly 512 bytes, but the packet
payload size must
127     be at least 580 bytes. The remaining bytes are used for the UDP, IP,
and Ethernet
128     headers and byte alignment requirements. */
129
130     status = nx_packet_pool_create(&server_pool, "TFTP Server Packet Pool",
NX_TFTP_PACKET_SIZE, pointer, 8192);
131     pointer = pointer + 8192;
132
133     /* Check for errors. */
134     if (status)
135         error_counter++;
136
137     /* Create the IP instance for the TFTP Server. */
138     status = nx_ip_create(&server_ip, "NetX Server IP Instance",
SERVER_ADDRESS, 0xFFFFFFFFUL,
139                               &server_pool, _nx_ram_network_driver,
pointer, 2048, 1);
140     pointer = pointer + 2048;
141
142     /* Check for errors. */
143     if (status)
144         error_counter++;
145
146     /* Enable ARP and supply ARP cache memory for IP Instance 0. */
147     status = nx_arp_enable(&server_ip, (void *) pointer, 1024);
148     pointer = pointer + 1024;
149
150     /* Check for errors. */
151     if (status)
152         error_counter++;
153
154     /* Enable UDP. */
155     status = nx_udp_enable(&server_ip);
156
157     /* Check for errors. */
158     if (status)
159         error_counter++;
160
161
162     /* Create the TFTP server. */
163     #ifdef USE_DUO
164     #if (IP_TYPE == 6)
165     #ifdef FEATURE_NX_IPV6
166     /* Specify the tftp server global address. */
167     server_ip_address.nxd_ip_version = NX_IP_VERSION_V6;
168     server_ip_address.nxd_ip_address.v6[0] = 0x20010db1;
169     server_ip_address.nxd_ip_address.v6[1] = 0xf101;
170     server_ip_address.nxd_ip_address.v6[2] = 0;
171     server_ip_address.nxd_ip_address.v6[3] = 0x102;
172     #endif
173     #else
174     server_ip_address.nxd_ip_version = NX_IP_VERSION_V4;

```

```

175     server_ip_address.nxd_ip_address.v4 = SERVER_ADDRESS;
176
177 #endif
178
179     status = nxd_tftp_server_create(&server, "TFTP Server Instance",
&server_ip, &ram_disk,
180                                     pointer, DEMO_STACK_SIZE, &server_pool);
181 #else
182     status = nx_tftp_server_create(&server, "TFTP Server Instance",
&server_ip, &ram_disk,
183                                     pointer, DEMO_STACK_SIZE, &server_pool);
184 #endif
185
186     pointer = pointer + DEMO_STACK_SIZE;
187
188     /* Check for errors for the server. */
189     if (status)
190         error_counter++;
191
192     /* Create a packet pool for the TFTP client. */
193
194     /* Note: The data portion of a packet is exactly 512 bytes, but the packet
payload size must
195         be at least 580 bytes. The remaining bytes are used for the UDP, IP,
and Ethernet
196         headers and byte alignment requirements. */
197
198     status = nx_packet_pool_create(&client_pool, "TFTP Client Packet Pool",
NX_TFTP_PACKET_SIZE, pointer, 8192);
199     pointer = pointer + 8192;
200
201     /* Create an IP instance for the TFTP client. */
202     status = nx_ip_create(&client_ip, "TFTP Client IP Instance",
CLIENT_ADDRESS, 0xFFFFFFFF0UL,
203                                     &client_pool,
_nx_ram_network_driver, pointer, 2048, 1);
204     pointer = pointer + 2048;
205
206     /* Enable ARP and supply ARP cache memory for IP Instance 1. */
207     status = nx_arp_enable(&client_ip, (void *) pointer, 1024);
208     pointer = pointer + 1024;
209
210     /* Enable UDP for client IP instance. */
211     status |= nx_udp_enable(&client_ip);
212     status |= nx_icmp_enable(&client_ip);
213
214     tx_thread_resume(&client_thread);
215 }
216
217 void server_thread_entry(ULONG thread_input)
218 {
219
220     UINT        status, running;
221     #if (IP_TYPE == 6)
222     #ifdef FEATURE_NX_IPV6
223     UINT        address_index;
224     UINT        iface_index;
225     #endif
226     #endif
227
228
229     /* Allow time for the network driver and NetX to get initialized. */
230     tx_thread_sleep(100);
231
232     #ifndef NX_TFTP_NO_FILEX
233
234     /* Format the RAM disk - the memory for the RAM disk was defined above.
*/
235     status = fx_media_format(&ram_disk,
236                             _fx_ram_driver,          /* Driver entry
*/
237                             ram_disk_memory,         /* RAM disk
memory pointer */
238                             ram_disk_sector_cache,   /* Media buffer
pointer */
239                             sizeof(ram_disk_sector_cache), /* Media buffer
size */
240                             "MY_RAM_DISK",           /* Volume Name
*/
241                             1,                       /* Number of FATs
*/

```

```

242      Entries          */          32,          /* Directory
243      */          0,          /* Hidden sectors
244      */          256,          /* Total sectors
245      */          128,          /* Sector size
246      cluster          */          1,          /* Sectors per
247      */          1,          /* Heads
248      track          */          1);          /* Sectors per
249
250      /* Check for errors. */
251      if (status != FX_SUCCESS)
252      {
253          return;
254      }
255
256      /* Open the RAM disk. */
257      status = fx_media_open(&ram_disk, "RAM DISK", _fx_ram_driver,
ram_disk_memory, ram_disk_sector_cache, sizeof(ram_disk_sector_cache));
258
259      /* Check for errors. */
260      if (status != FX_SUCCESS)
261      {
262          return;
263      }
264
265      #endif /* NX_TFTP_NO_FILEX */
266
267      #if (IP_TYPE == 6)
268      #ifdef FEATURE_NX_IPV6
269          /* Enable ICMPv6 services. */
270          status |= nxd_icmp_enable(&server_ip);
271          if (status != NX_SUCCESS)
272          {
273              return;
274          }
275
276          /* Enable IPv6 services for the server. */
277          status = nxd_ipv6_enable(&server_ip);
278          if (status != NX_SUCCESS)
279          {
280              return;
281          }
282
283          /* This assumes the primary interface. See the NetX Duo
284             User Guide for more information on address configuration. */
285          iface_index = 0;
286          status = nxd_ipv6_address_set(&server_ip, iface_index, NX_NULL, 10,
&address_index);
287          status += nxd_ipv6_address_set(&server_ip, iface_index, &server_ip_address,
64, &address_index);
288
289          if (status != NX_SUCCESS)
290          {
291              return;
292          }
293
294          /* wait for DAD to validate the address. */
295          tx_thread_sleep(500);
296      #endif
297
298      #endif /* IP_TYPE == 6 */
299
300      /* Start the NetX TFTP server. */
301      #ifdef USE_DUO
302          status = nxd_tftp_server_start(&server);
303      #else
304          status = nx_tftp_server_start(&server);
305      #endif
306
307      /* Check for errors. */
308      if (status)
309      {
310          error_counter++;
311          return;
312

```



```

313     }
314
315     /* Run for a while */
316     running = NX_TRUE;
317     while(running)
318         tx_thread_sleep(200);
319
320 #ifdef USE_DUO
321     nxd_tftp_server_delete(&server);
322 #else
323     nx_tftp_server_delete(&server);
324 #endif
325
326 }
327
328 /* Define the TFTP client thread. */
329 void client_thread_entry(ULONG thread_input)
330 {
331     NX_PACKET *my_packet;
332     UINT status;
333     UINT all_done = NX_FALSE;
334     #if (IP_TYPE == 6)
335     #ifdef FEATURE_NX_IPV6
336         address_index;
337         iface_index;
338     #endif
339     #endif
340
341     /* Allow time for the network driver and NetX to get initialized. */
342     tx_thread_sleep(100);
343
344     #if (IP_TYPE == 6)
345     #ifdef FEATURE_NX_IPV6
346         /* Enable ECMPv6 services for the client. */
347         status = nxd_icmp_enable(&client_ip);
348         if (status != NX_SUCCESS)
349         {
350             return;
351         }
352
353         /* Enable IPv6 services for the client. */
354         status = nxd_ipv6_enable(&client_ip);
355         if (status != NX_SUCCESS)
356         {
357             return;
358         }
359
360         /* Set the Client IPv6 address */
361         client_ip_address.nxd_ip_version = NX_IP_VERSION_V6;
362         client_ip_address.nxd_ip_address.v6[0] = 0x20010db1;
363         client_ip_address.nxd_ip_address.v6[1] = 0xf101;
364         client_ip_address.nxd_ip_address.v6[2] = 0;
365         client_ip_address.nxd_ip_address.v6[3] = 0x101;
366
367         /* This assumes the primary interface. See the NetX Duo
368            User Guide for more information on address configuration. */
369         iface_index = 0;
370         status = nxd_ipv6_address_set(&client_ip, iface_index, NX_NULL, 10,
371 &address_index);
372         status += nxd_ipv6_address_set(&client_ip, iface_index, &client_ip_address,
373 64, &address_index);
374         if (status != NX_SUCCESS)
375         {
376             return;
377         }
378
379         /* wait for the link local and global addresses to be validated. */
380         tx_thread_sleep(500);
381     #endif
382     #endif /*(IP_TYPE == 6) */
383
384     /* The TFTP services used below include the NetX equivalent service which
385        will work with

```

```

390         NetX Duo TFTP. However, it is recommended for developers to port their
391         applications to the newer services that take the NXD_ADDRESS type and support both
392         IPv4 and IPv6 communication.
393     */
394
395     /* Create a TFTP client. */
396 #ifdef USE_DUO
397     status = nxd_tftp_client_create(&client, "TFTP Client", &client_ip,
&client_pool, IP_TYPE);
398 #else
399     status = nx_tftp_client_create(&client, "TFTP Client", &client_ip,
&client_pool);
400 #endif
401
402     /* Check status. */
403     if (status)
404         return;
405
406     /* Open a TFTP file for writing. */
407 #ifdef USE_DUO
408     status = nxd_tftp_client_file_open(&client, "test.txt",
&server_ip_address, NX_TFTP_OPEN_FOR_WRITE, 100, IP_TYPE);
409 #else
410     status = nx_tftp_client_file_open(&client, "test.txt", SERVER_ADDRESS,
NX_TFTP_OPEN_FOR_WRITE, 100);
411 #endif
412
413     /* Check status. */
414     if (status)
415         return;
416
417     /* Allocate a TFTP packet. */
418 #ifdef USE_DUO
419     status = nxd_tftp_client_packet_allocate(&client_pool, &my_packet, 100,
IP_TYPE);
420 #else
421     status = nx_tftp_client_packet_allocate(&client_pool, &my_packet, 100);
422 #endif
423     /* Check status. */
424     if (status)
425         error_counter++;
426
427     /* Write ABCs into the packet payload! */
428     memcpy(my_packet -> nx_packet_prepend_ptr, "ABCDEFGHIJKLMNOPQRSTUVWXYZ ",
28);
429
430     /* Adjust the write pointer. */
431     my_packet -> nx_packet_length = 28;
432     my_packet -> nx_packet_append_ptr = my_packet -> nx_packet_prepend_ptr +
28;
433
434     /* Write this packet to the file via TFTP. */
435 #ifdef USE_DUO
436     status = nxd_tftp_client_file_write(&client, my_packet, 100, IP_TYPE);
437 #else
438     status = nx_tftp_client_file_write(&client, my_packet, 100);
439 #endif
440
441     /* Check status. */
442     if (status)
443         error_counter++;
444
445     /* Close this file. */
446 #ifdef USE_DUO
447     status = nxd_tftp_client_file_close(&client, IP_TYPE);
448 #else
449     status = nx_tftp_client_file_close(&client);
450 #endif
451
452     /* Check status. */
453     if (status)
454         error_counter++;
455
456     /* Open the same file for reading. */
457 #ifdef USE_DUO
458     status = nxd_tftp_client_file_open(&client, "test.txt",
&server_ip_address, NX_TFTP_OPEN_FOR_READ, 100, IP_TYPE);
459 #else

```

```

460     status = nx_tftp_client_file_open(&client, "test.txt", SERVER_ADDRESS,
NX_TFTP_OPEN_FOR_READ, 100);
461 #endif
462
463     /* Check status. */
464     if (status)
465         error_counter++;
466     do
467     {
468
469         /* Read the file back. */
470 #ifdef USE_DUO
471         status = nxd_tftp_client_file_read(&client, &my_packet, 100, IP_TYPE);
472 #else
473         status = nx_tftp_client_file_read(&client, &my_packet, 100);
474 #endif
475         /* Check for retransmission/dropped packet error. Benign. Try again...
*/
476         if (status == NX_TFTP_INVALID_BLOCK_NUMBER)
477         {
478
479             continue;
480         }
481         else if (status == NX_TFTP_END_OF_FILE)
482         {
483
484             /* All done. */
485             all_done = NX_TRUE;
486         }
487         else if (status != NX_SUCCESS)
488         {
489
490             /* Internal error, invalid packet or error on read. */
491             break;
492         }
493
494
495         /* Do something with the packet data and release when done. */
496         nx_packet_data_retrieve(my_packet, buffer, &data_length);
497         buffer[data_length] = 0;
498         printf("Receive data: %s\n", buffer);
499
500         printf("release packet in demo.\n");
501
502         nx_packet_release(my_packet);
503     } while (all_done == NX_FALSE);
504
505     /* Close the file again. */
506 #ifdef USE_DUO
507     status = nxd_tftp_client_file_close(&client, IP_TYPE);
508 #else
509     status = nx_tftp_client_file_close(&client);
510 #endif
511 #endif
512
513     /* Check status. */
514     if (status)
515         error_counter++;
516
517     /* Delete the client. */
518 #ifdef USE_DUO
519     status = nxd_tftp_client_delete(&client);
520 #else
521     status = nx_tftp_client_delete(&client);
522 #endif
523
524     /* Check status. */
525     if (status)
526         error_counter++;
527
528     return;
529 }

```

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	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_SERVER_TIME_SLICE	The time slice for the TFTP Server to run before yielding to other threads of the same priority. The default value is 2.
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.

NX_TFTP_FRAGMENT_OPTION

Fragment enable for TFTP UDP requests. By default, this value is NX_DONT_FRAGMENT to disable TFTP UDP fragmenting.

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.

NX_TFTP_SOURCE_PORT

This option allows a TFTP Client application to specify the TFTP Client UDP socket port. It is defaulted to NX_ANY_PORT.

NX_TFTP_SERVER_RETRANSMIT_ENABLE

Enables the TFTP server's timer to check each TFTP client session with for recent activity (either an ACK or data packet). When the session timeout expires after the maximum number of times, it is assumed the connection was lost. The Server clears the Client request, closes any open files and makes the connection request available for the next Client. The default setting is disabled.

NX_TFTP_SERVER_TIMEOUT_PERIOD

Specifies the interval when the TFTP server timer entry function checks Client connections for receiving any packets. The default value is 20 (timer ticks).

NX_TFTP_SERVER_RETRANSMIT_TIMEOUT

This is the timeout for receiving a valid ACK or data packet from

the Client. The default value is 200 (timer ticks).

NX_TFTP_SERVER_MAX_RETRIES Specifies the maximum number of times the Client session retransmit timeout is renewed. Thereafter, the session is closed by the Server.

NX_TFTP_MAX_CLIENT_RETRANSMITS Specifies the maximum number of times the Server receives a duplicate ACK or data packet from the Client (which it drops) without sending an error message to the Client and closing the session. Has no effect if **NX_TFTP_SERVER_RETRANSMIT_ENABLE** is defined.

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.

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_set_interface
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

Note: The IPv4 equivalents of all the services listed above are available in NetX Duo TFTP Client and Server e.g. *nx_tftp_server_create* and *nx_tftp_client_file_open*. Only the 'Duo' API descriptions, e.g. services beginning with *nxd_*, are provided in the following pages. Where an NXD_ADDRESS * input is specified, the IPv4 equivalent API calls for ULONG input. Otherwise there is no difference in using the API.

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.
NX_TFTP_INVALID_IP_VERSION	(0x0C)	Invalid or unsupported IP version
NX_TFTP_INVALID_SERVER_ADDRESS	(0x08)	Invalid Server IP address received
NX_TFTP_NO_ACK_RECEIVED	(0x09)	Server ACK not received
NX_PTR_ERROR	(0x16)	Invalid IP, pool, or TFTP pointer.
NX_INVALID_PARAMETERS	(0x4D)	Invalid non pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

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. */
```

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. */
```

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. */
```

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.
NX_PTR_ERROR	(0x16)	Invalid pointer input.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.
NX_INVALID_PARAMETERS	(0x4D)	Invalid non pointer input

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. */
```

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

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_TFTP_INVALID_SERVER_ADDRESS	(0x08)	Invalid Server IP received
NX_TFTP_CODE_ERROR	(0x05)	Received error code
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. */
```

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

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_TFTP_INVALID_IP_VERSION	(0x0C)	Invalid IP version
NX_TFTP_INVALID_SERVER_ADDRESS	(0x08)	Invalid Server IP received
NX_TFTP_CODE_ERROR	(0x05)	Received error code
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
NX_INVALID_PARAMETERS	(0x4D)	Invalid non pointer input

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. */
```

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 0xFFFFFFFF) 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 (1-0xFFFFFFFF) 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
NX_TFTP_END_OF_FILE	(0xC5)	End of file detected (not an error).
NX_TFTP_INVALID_IP_VERSION	(0x0C)	Invalid IP version
NX_TFTP_CODE_ERROR	(0x05)	Received error code
NX_TFTP_FAILED	(0xC2)	Unknown TFTP code received
NX_TFTP_INVALID_BLOCK_NUMBER	(0x0A)	Invalid block number received
NX_PTR_ERROR	(0x16)	Invalid pointer input.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service
NX_INVALID_PARAMETERS	(0x4D)	Invalid non pointer input

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". */

```

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 0xFFFFFFFF) 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 (1-0xFFFFFFFF) specifies the maximum number of timer-ticks to stay suspended while waiting for the TFTP server to send an ACK for the write request.
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
NX_TFTP_INVALID_IP_VERSION	(0x0C)	Invalid IP version
NX_INVALID_TFTP_SERVER_ADDRESS	(0x08)	Invalid server address received
NX_TFTP_CODE_ERROR	(0x05)	Received error code
NX_PTR_ERROR	(0x16)	Invalid pointer input.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service
NX_INVALID_PARAMETERS	(0x4D)	Invalid non pointer input

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". */
```

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: <div style="margin-left: 40px;"> timeout value (0x00000001 through 0xFFFFFFFF) TX_WAIT_FOREVER (0xFFFFFFFF) Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until 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. </div>
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_PTR_ERROR	(0x16)	Invalid pointer input.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service
NX_INVALID_PARAMETERS	(0x4D)	Invalid non pointer input

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. */
```

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
	(0x0B)	Invalid interface input
NX_PTR_ERROR	(0x16)	Invalid pointer input.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service
NX_TFTP_INVALID_INTERFACE	(0x0B)	Invalid interface input

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. */
```


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
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. */
```

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. */
```

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. */
```

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. */
```

Trivial File Transfer Protocol (TFTP) for NetX Duo
User Guide

Publication Date: Rev.5.12 Nov 8, 2018

Published by: Renesas Electronics Corporation



SALES OFFICES

Renesas Electronics Corporation

<http://www.renesas.com>

Refer to "<http://www.renesas.com/>" for the latest and detailed information.

Renesas Electronics Corporation

TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

Renesas Electronics America Inc.

1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.

Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited

9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3

Tel: +1-905-237-2004

Renesas Electronics Europe Limited

Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K

Tel: +44-1628-651-700

Renesas Electronics Europe GmbH

Arcadiastrasse 10, 40472 Düsseldorf, Germany

Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.

Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China

Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.

Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China

Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited

Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong

Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.

13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan

Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.

80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949

Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.

Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia

Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.

No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India

Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.

17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea

Tel: +82-2-558-3737, Fax: +82-2-558-5338

Trivial File Transfer Protocol (TFTP) for NetX Duo User Guide