

NetX™

Trivial File Transfer Protocol (TFTP)

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 TFTP for the Renesas Synergy platform, please use the following information.

TFTP Multi-Thread Support

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

Installation

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



Trivial File Transfer Protocol (TFTP)

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 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 TFTP	8
Product Distribution.....	8
TFTP Installation	8
Using TFTP	8
Small Example System	9
Configuration Options	13
Chapter 3 Description of TFTP Services	16
nx_tftp_client_create.....	18
nx_tftp_client_delete.....	19
nx_tftp_client_error_info_get	20
nx_tftp_client_file_close.....	21
nx_tftp_client_file_open	22
nx_tftp_client_file_read.....	24
nx_tftp_client_file_write.....	26
nx_tftp_client_packet_allocate	28
nx_tftp_client_set_interface.....	30
nx_tftp_server_create.....	31
nx_tftp_server_delete.....	33
nx_tftp_server_start	34
nx_tftp_server_stop.....	35

Chapter 1

Introduction to TFTP

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

TFTP Requirements

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

The TFTP Server portion of the NetX TFTP package has several additional requirements. First, it requires complete access to UDP *well known port 69* for handling all client TFTP requests. The TFTP Server is also designed for use with the FileX embedded file system. If FileX is not available, the user may port the portions of FileX used to their own environment. This is discussed in later sections of this guide.

TFTP File Names

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

TFTP Messages

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

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

The TFTP Server utilizes the well-known UDP port 69 to field client requests. TFTP Clients may use any available UDP port. Data packets are fixed at 512 bytes, until the last packet. A packet containing fewer than 512 bytes signals the end of file. The general sequence of events is as follows:

TFTP Read File Requests:

1. Client Issues "Open For Read" request with the File Name and waits for a packet from Server.
2. Server sends the first 512 bytes of the file.
3. Client receives data, sends ACK, and waits for the next packet if the last packet had 512 bytes.
4. The sequence ends when a packet containing fewer than 512 bytes is received.

TFTP Write Requests:

1. Client Issues "Open for Write" request with the File Name and waits for an ACK with a block number of 0 from the Server.
2. When the Server is ready to write the file, it sends an ACK with a block number of zero.
3. Client sends the first 512 bytes of the file to the Server and waits for an ACK.
4. Server sends ACK after the bytes are written.
5. The sequence ends when the Client completes writing a packet containing fewer than 512 bytes.

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

TFTP RFCs

NetX TFTP is compliant with RFC1350 and related RFCs.

Chapter 2

Installation and Use of TFTP

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

Product Distribution

TFTP for NetX is shipped on a single CD-ROM compatible disk. The package includes two source files and a PDF file that contains this document, as follows:

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

TFTP Installation

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

Using TFTP

To run a TFTP application, the application code must include the header files after it includes *tx_api.h*, *fx_api.h*, and *nx_api.h*, in order to use ThreadX, FileX, and NetX, respectively. Once the header files are included, the application code is then able to make the TFTP function calls specified later in this guide. The application must also include *nx_tftp_client.c* and *nx_tftp_server.c* in the build process. These files must be compiled in the same manner as other application files and its object form must be linked along with the files of the application. This is all that is required to use NetX TFTP.

Note that since TFTP utilizes NetX UDP services, UDP must be enabled with the `nx_udp_enable` call prior to using TFTP.

Small Example System

An example of how easy it is to use NetX TFTP is described in Figure 1.1 that appears below. In this example, the TFTP include file `nx_tftp_client.h` and `nx_tftp_server.h` are brought at line 11 and 12. Next, the TFTP server is created in “`tx_application_define`” at line 153. Note that the TFTP Server control block “`server`” was defined as a global variable at line 37 previously. After successful creation, a TFTP Server is started at line 233. At line 272 the TFTP Client is created. And finally, the client writes the file at line 299 and reads the file back at line 322.

```

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 #include "tx_api.h"
10 #include "nx_api.h"
11 #include "nx_tftp_client.h"
12 #include "nx_tftp_server.h"
13 #ifndef NX_TFTP_NO_FILEX
14 #include "fx_api.h"
15 #endif
16
17 #define DEMO_STACK_SIZE 4096
18
19 /* To use another file storage utility define this symbol:
20 */
21 #define NX_TFTP_NO_FILEX
22
23
24 /* Define the ThreadX, NetX, and FileX object control blocks... */
25
26 TX_THREAD server_thread;
27 TX_THREAD client_thread;
28 NX_PACKET_POOL server_pool;
29 NX_IP server_ip;
30 NX_PACKET_POOL client_pool;
31 NX_IP client_ip;
32 FX_MEDIA ram_disk;
33
34 /* Define the NetX TFTP object control blocks. */
35
36 NX_TFTP_CLIENT client;
37 NX_TFTP_SERVER server;
38
39 /* Define the application global variables */
40
41 #define CLIENT_ADDRESS IP_ADDRESS(1, 2, 3, 5)
42 #define SERVER_ADDRESS IP_ADDRESS(1, 2, 3, 4)
43
44
45 UINT error_counter = 0;
46
47 /* Define buffer used in the demo application. */
48 UCHAR buffer[255];
49 ULONG data_length;
50
51
52 /* Define the memory area for the FileX RAM disk. */
53 #ifndef NX_TFTP_NO_FILEX
54 UCHAR ram_disk_memory[32000];
55 UCHAR ram_disk_sector_cache[512];
56 #endif
57
58
59 /* Define function prototypes. */
60
61 VOID _fx_ram_driver(FX_MEDIA *media_ptr);
62 VOID _nx_ram_network_driver(NX_IP_DRIVER *driver_req_ptr);
63 VOID client_thread_entry(ULONG thread_input);
64 VOID server_thread_entry(ULONG thread_input);
65
66
67 /* Define main entry point. */
68
69 int main()
70 {
71     /* Enter the ThreadX kernel. */
72     tx_kernel_enter();
73 }
74
75
76 /* Define what the initial system looks like. */
77
78 void tx_application_define(void *first_unused_memory)
79 {
80     UINT status;
81     UCHAR *pointer;
82
83     /* Setup the working pointer. */
84     pointer = (UCHAR *) first_unused_memory;
85
86     /* ... (rest of the application logic) ... */
87
88 }
89

```

```

90  /* Create the main TFTP server thread. */
91  status = tx_thread_create(&server_thread, "TFTP Server Thread", server_thread_entry, 0,
92                          pointer, DEMO_STACK_SIZE,
93                          4, 4, TX_NO_TIME_SLICE, TX_AUTO_START);
94
95  pointer += DEMO_STACK_SIZE ;
96
97  /* Check for errors. */
98  if (status)
99      error_counter++;
100
101
102  /* Create the main TFTP client thread at a slightly lower priority. */
103  status = tx_thread_create(&client_thread, "TFTP Client Thread", client_thread_entry, 0,
104                          pointer, DEMO_STACK_SIZE,
105                          5, 5, TX_NO_TIME_SLICE, TX_DONT_START);
106
107  pointer += DEMO_STACK_SIZE ;
108
109  /* Check for errors. */
110  if (status)
111      error_counter++;
112
113  /* Initialize the NetX system. */
114  nx_system_initialize();
115
116  /* Note: The data portion of a packet is exactly 512 bytes, but the packet payload size must
117  be at least 580 bytes. The remaining bytes are used for the UDP, IP, and Ethernet
118  headers and byte alignment requirements. */
119
120  status = nx_packet_pool_create(&server_pool, "TFTP Server Packet Pool", NX_TFTP_PACKET_SIZE, pointer, 8192);
121  pointer = pointer + 8192;
122
123  /* Check for errors. */
124  if (status)
125      error_counter++;
126
127  /* Create the IP instance for the TFTP Server. */
128  status = nx_ip_create(&server_ip, "NetX Server IP Instance", SERVER_ADDRESS, 0xFFFFF00UL,
129                      &server_pool, _nx_ram_network_driver, pointer, 2048, 1);
130  pointer = pointer + 2048;
131
132  /* Check for errors. */
133  if (status)
134      error_counter++;
135
136  /* Enable ARP and supply ARP cache memory for IP Instance 0. */
137  status = nx_arp_enable(&server_ip, (void *) pointer, 1024);
138  pointer = pointer + 1024;
139
140  /* Check for errors. */
141  if (status)
142      error_counter++;
143
144  /* Enable UDP. */
145  status = nx_udp_enable(&server_ip);
146
147  /* Check for errors. */
148  if (status)
149      error_counter++;
150
151
152  /* Create the TFTP server. */
153  status = nx_tftp_server_create(&server, "TFTP Server Instance", &server_ip, &ram_disk,
154                              pointer, DEMO_STACK_SIZE, &server_pool);
155
156  pointer = pointer + DEMO_STACK_SIZE;
157
158  /* Check for errors for the server. */
159  if (status)
160      error_counter++;
161
162  /* Create a packet pool for the TFTP client. */
163
164  /* Note: The data portion of a packet is exactly 512 bytes, but the packet payload size must
165  be at least 580 bytes. The remaining bytes are used for the UDP, IP, and Ethernet
166  headers and byte alignment requirements. */
167
168  status = nx_packet_pool_create(&client_pool, "TFTP Client Packet Pool", NX_TFTP_PACKET_SIZE, pointer, 8192);
169  pointer = pointer + 8192;
170
171  /* Create an IP instance for the TFTP client. */
172  status = nx_ip_create(&client_ip, "TFTP Client IP Instance", CLIENT_ADDRESS, 0xFFFFF00UL,
173                      &client_pool, _nx_ram_network_driver, pointer, 2048, 1);
174  pointer = pointer + 2048;
175
176  /* Enable ARP and supply ARP cache memory for IP Instance 1. */
177  status = nx_arp_enable(&client_ip, (void *) pointer, 1024);
178  pointer = pointer + 1024;
179
180  /* Enable UDP for client IP instance. */
181  status |= nx_udp_enable(&client_ip);
182  status |= nx_icmp_enable(&client_ip);
183
184  tx_thread_resume(&client_thread);
185 }
186
187 void server_thread_entry(ULONG thread_input)
188 {
189
190  UINT    status, running;
191
192
193  /* Allow time for the network driver and NetX to get initialized. */
194  tx_thread_sleep(100);
195
196  #ifndef NX_TFTP_NO_FILEX
197
198  /* Format the RAM disk - the memory for the RAM disk was defined above. */
199  status = fx_media_format(&ram_disk,
200                          _fx_ram_driver,          /* Driver entry */
201                          ram_disk_memory,         /* RAM disk memory pointer */
202                          ram_disk_sector_cache,    /* Media buffer pointer */
203                          sizeof(ram_disk_sector_cache), /* Media buffer size */
204                          "MY_RAM_DISK",           /* Volume Name */
205                          1,                        /* Number of FATS */
206                          32,                       /* Directory Entries */
207                          0,                        /* Hidden sectors */
208                          256,                      /* Total sectors */
209                          128,                     /* Sector size */
210                          1,                        /* Sectors per cluster */
211                          1,                        /* Heads */
212                          1);                      /* Sectors per track */
213
214  /* Check for errors. */
215  if (status != FX_SUCCESS)
216  {
217      return;
218  }

```

```

219
220 /* Open the RAM disk. */
221 status = fx_media_open(&ram_disk, "RAM DISK", _fx_ram_driver, ram_disk_memory, ram_disk_sector_cache,
222 sizeof(ram_disk_sector_cache));
223
224 /* Check for errors. */
225 if (status != FX_SUCCESS)
226 {
227     return;
228 }
229 #endif /* NX_TFTP_NO_FILEX */
230
231 /* Start the NetX TFTP server. */
232 status = nx_tftp_server_start(&server);
233
234 /* Check for errors. */
235 if (status)
236 {
237     error_counter++;
238     return;
239 }
240
241 /* Run for a while */
242 running = NX_TRUE;
243 while(running)
244     tx_thread_sleep(200);
245
246 nx_tftp_server_delete(&server);
247
248 }
249
250 }
251
252 /* Define the TFTP client thread. */
253
254 void client_thread_entry(ULONG thread_input)
255 {
256     NX_PACKET *my_packet;
257     UINT status;
258     UINT all_done = NX_FALSE;
259
260     /* Allow time for the network driver and NetX to get initialized. */
261     tx_thread_sleep(100);
262
263     /* The TFTP services used below include the NetX equivalent service which will work with
264     NetX TFTP.
265     */
266
267     /* Create a TFTP client. */
268     status = nx_tftp_client_create(&client, "TFTP client", &client_ip, &client_pool);
269
270     /* Check status. */
271     if (status)
272     {
273         return;
274     }
275
276     /* Open a TFTP file for writing. */
277     status = nx_tftp_client_file_open(&client, "test.txt", SERVER_ADDRESS, NX_TFTP_OPEN_FOR_WRITE, 100);
278
279     /* Check status. */
280     if (status)
281     {
282         return;
283     }
284
285     /* Allocate a TFTP packet. */
286     status = nx_tftp_client_packet_allocate(&client_pool, &my_packet, 100);
287     /* Check status. */
288     if (status)
289     {
290         error_counter++;
291     }
292
293     /* Write ABCs into the packet payload! */
294     memcpy(my_packet->nx_packet_prepend_ptr, "ABCDEFGHJKLMNOPQRSTUVWXYZ ", 28);
295
296     /* Adjust the write pointer. */
297     my_packet->nx_packet_length = 28;
298     my_packet->nx_packet_append_ptr = my_packet->nx_packet_prepend_ptr + 28;
299
300     /* Write this packet to the file via TFTP. */
301     status = nx_tftp_client_file_write(&client, my_packet, 100);
302
303     /* Check status. */
304     if (status)
305     {
306         error_counter++;
307     }
308
309     /* Close this file. */
310     status = nx_tftp_client_file_close(&client);
311
312     /* Check status. */
313     if (status)
314     {
315         error_counter++;
316     }
317
318     /* Open the same file for reading. */
319     status = nx_tftp_client_file_open(&client, "test.txt", SERVER_ADDRESS, NX_TFTP_OPEN_FOR_READ, 100);
320
321     /* Check status. */
322     if (status)
323     {
324         error_counter++;
325     }
326
327     do
328     {
329         /* Read the file back. */
330         status = nx_tftp_client_file_read(&client, &my_packet, 100);
331         /* Check for retransmission/dropped packet error. Benign. Try again... */
332         if (status == NX_TFTP_INVALID_BLOCK_NUMBER)
333         {
334             continue;
335         }
336         else if (status == NX_TFTP_END_OF_FILE)
337         {
338             /* All done. */
339             all_done = NX_TRUE;
340         }
341         else if (status != NX_SUCCESS)
342         {
343             /* Internal error, invalid packet or error on read. */
344             break;
345         }
346
347         /* Do something with the packet data and release when done. */
348         nx_packet_data_retrieve(my_packet, buffer, &data_length);
349         buffer[data_length] = 0;
350         printf("Receive data: %s\n", buffer);
351     }

```

```
347         printf("release packet in demo.\n");
348     349
349         nx_packet_release(my_packet);
350     351
351     } while (all_done == NX_FALSE);
352 353
353     /* Close the file again. */
354     status = nx_tftp_client_file_close(&client);
355 356
356     /* Check status. */
357     if (status)
358         error_counter++;
359 360
360     /* Delete the client. */
361     status = nx_tftp_client_delete(&client);
362 363
363     /* Check status. */
364     if (status)
365         error_counter++;
366 367
367     return;
368 369 }
```

Figure 1.1 Example of TFTP use with NetX

Configuration Options

There are several configuration options for building TFTP for NetX. The following list describes each in detail. Unless otherwise specified, these options are found in *nx_tftp_client.h* and *nx_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 TFTP services (listed below) in alphabetic order.

In the “Return Values” section in the following API descriptions, values in **BOLD** are not affected by the **NX_DISABLE_ERROR_CHECKING** define that is used to disable API error checking, while non-bold values are completely disabled.

`nx_tftp_client_create`
Create a TFTP client instance

`nx_tftp_client_delete`
Delete a TFTP client instance

`nx_tftp_client_error_info_get`
Get client error information

`nx_tftp_client_file_close`
Close client file

`nx_tftp_client_file_open`
Open client file

`nx_tftp_client_file_read`
Read a block from client file

`nx_tftp_client_file_write`
Write block to client file

`nx_tftp_client_packet_allocate`
Allocate packet for client file write

`nx_tftp_client_set_interface`
Set the physical interface for TFTP requests

`nx_tftp_server_create`
Create TFTP server

`nx_tftp_server_delete`
Delete TFTP server

`nx_tftp_server_start`
Start TFTP Server

`nx_tftp_server_stop`
Stop TFTP Server

nx_tftp_client_create

Create a TFTP client instance

Prototype

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

Description

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

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

Input Parameters

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

Return Values

NX_SUCCESS	(0x00)	Successful TFTP create.
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_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Initialization and Threads

Example

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

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

nx_tftp_client_delete

Delete a TFTP client instance

Prototype

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

Description

This service deletes a previously created TFTP client instance.

Input Parameters

tftp_client_ptr	Pointer to previously created TFTP client instance.
------------------------	---

Return Values

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

Allowed From

Threads

Example

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

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

nx_tftp_client_error_info_get

Get client error information

Prototype

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

Description

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

Input Parameters

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

Return Values

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

Allowed From

Threads

Example

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

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

nx_tftp_client_file_close

Close client file

Prototype

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

Description

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

Input Parameters

tftp_client_ptr	Pointer to previously created TFTP client instance.
------------------------	---

Return Values

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

Allowed From

Threads

Example

```
/* Close the previously opened file associated with "my_client". */
status = nx_tftp_client_file_close(&my_tftp_client);
/* If status is NX_SUCCESS the TFTP file is closed. */
```


nx_tftp_client_file_open

Open client file

Prototype

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

Description

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

Input Parameters

tftp_client_ptr	Pointer to TFTP control block.
------------------------	--------------------------------

file_name	ASCII file name, NULL-terminated and with appropriate path information.
------------------	---

server_ip_address IP address of TFTP Server.

open_type	Type of open request, either:
------------------	-------------------------------

NX_TFTP_OPEN_FOR_READ (0x01)
NX_TFTP_OPEN_FOR_WRITE (0x02)

wait_option	Defines how long the service will wait for the TFTP client file open. The wait options are defined as follows:
--------------------	--

timeout value (0x00000001 through 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.

Return Values

NX_SUCCESS	(0x00)	Successful TFTP client file open
NX_TFTP_NOT_CLOSED	(0xC3)	Client already has file open
NX_INVALID_TFTP_SERVER_ADDRESS	(0x08)	Invalid server address received
NX_TFTP_NO_ACK_RECEIVED	(0x09)	No ACK received from server
NX_TFTP_INVALID_SERVER_ADDRESS	(0x08)	Invalid Server IP received
NX_TFTP_CODE_ERROR	(0x05)	Received error code from Server
NX_PTR_ERROR	(0x16)	Invalid TFTP pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service
NX_IP_ADDRESS_ERROR	(0x21)	Invalid TFTP Server IP address
NX_OPTION_ERROR	(0x0a)	Invalid open type

Allowed From

Threads

Example

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

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

nx_tftp_client_file_read

Read a block from client file

Prototype

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

Description

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

Input Parameters

tftp_client_ptr	Pointer to TFTP client control block.
packet_ptr	Destination for packet containing the block read from the file.
wait_option	Defines how long the service will wait for the read to complete. The wait options are defined as follows: timeout value (0x00000001 through 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.

Return Values

NX_SUCCESS	(0x00)	Successful 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_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**Allowed From**

Threads

Example

```

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

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

```

nx_tftp_client_file_write

Write block to client file

Prototype

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

Description

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

Input Parameters

tftp_client_ptr	Pointer to TFTP client control block.
packet_ptr	Packet containing the block to write to the file.
wait_option	Defines how long the service will wait for the write to complete. The wait options are defined as follows: timeout value (0x00000001 through 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.

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_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

Allowed From

Threads

Example

```
/* write a block to the previously opened file of "my_client". */
status = nx_tftp_client_file_write(&my_tftp_client, packet_ptr, 200);
/* If status is NX_SUCCESS the block in the payload of "packet_ptr" was
   written to the TFPT file opened by "my_client". */
```

nx_tftp_client_packet_allocate

Allocate packet for client file write

Prototype

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

Description

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

Input Parameters

pool_ptr	Pointer to packet pool.
packet_ptr	Destination for pointer to allocated packet.
wait_option	Defines how long the service will wait for the packet allocate to complete. The wait options are defined as follows:
timeout value	(0x00000001 through 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.

Return Values

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

Allowed From

Threads

Example

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


nx_tftp_client_set_interface

Set physical interface for TFTP requests

Prototype

```
UINT nx_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.

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 = nx_tftp_client_set_interface(&client, 0);

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

nx_tftp_server_create

Create TFTP server

Prototype

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

Description

This service creates a TFTP server that responds to TFTP client requests on port 69. The server must be started by a subsequent call to *nx_tftp_server_start*.

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

Input Parameters

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

Return Values

NX_SUCCESS	(0x00)	Successful TFTP server create
-------------------	--------	-------------------------------

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

NX_TFTP_POOL_ERROR	(0xC6)	Packet pool has packet size of less than 560 bytes
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Initialization, Threads

Example

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

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

nx_tftp_server_delete

Delete TFTP server

Prototype

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

Description

This service deletes a previously created TFTP server.

Input Parameters

tftp_server_ptr Pointer to TFTP server control block.

Return Values

NX_SUCCESS	(0x00)	Successful Server delete
NX_PTR_ERROR	(0x16)	Invalid Server pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller

Allowed From

Threads

Example

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

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

nx_tftp_server_start

Start TFTP server

Prototype

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

Description

This service starts the previously created TFTP server.

Input Parameters

tftp_server_ptr	Pointer to TFTP server control block.
------------------------	---------------------------------------

Return Values

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

Allowed From

Initialization, threads

Example

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

nx_tftp_server_stop

Stop TFTP server

Prototype

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

Description

This service stops the previously created TFTP server.

Input Parameters

tftp_server_ptr Pointer to TFTP server control block.

Return Values

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

Allowed From

Threads

Example

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

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

NetX Trivial File Transfer Protocol (TFTP) User Guide

Publication Date: Rev.5.10 Nov 7, 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

NetX Trivial File Transfer Protocol (TFTP) User Guide