

NetX™

Telnet Protocol (Telnet)

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

Multi-Thread Support

Page 8: Multi-thread support has not been tested for SSP v1.5.0.

Installation

Page 9: If you are using Renesas Synergy SSP and the e² studio ISDE, Telnet will already be installed. You can ignore the NetX Telnet Installation section.



Telnet Protocol (Telnet)

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: 000-1055
Revision 5.11

Contents

| | |
|---|----|
| Chapter 1 Introduction to Telnet..... | 4 |
| Telnet Requirements | 4 |
| Telnet Constraints | 4 |
| Telnet Communication | 4 |
| Telnet Authentication..... | 5 |
| Telnet New Connection Callback | 5 |
| Telnet Receive Data Callback | 5 |
| Telnet End Connection Callback | 6 |
| Telnet Option Negotiation..... | 7 |
| Telnet Multi-Thread Support..... | 8 |
| Telnet RFCs | 8 |
| Chapter 2 Installation and Use of Telnet..... | 9 |
| Product Distribution | 9 |
| Telnet Installation | 9 |
| Using Telnet | 9 |
| Small Example System | 10 |
| Configuration Options..... | 15 |
| Chapter 3 Description of Telnet Services..... | 18 |
| nx_telnet_client_connect..... | 20 |
| nx_telnet_client_create | 22 |
| nx_telnet_client_delete..... | 23 |
| nx_telnet_client_disconnect | 24 |
| nx_telnet_client_packet_receive | 26 |
| nx_telnet_client_packet_send | 28 |
| nx_telnet_server_create..... | 30 |
| nx_telnet_server_delete | 32 |
| nx_telnet_server_disconnect..... | 33 |
| nx_telnet_server_get_open_connection_count..... | 34 |
| nx_telnet_server_packet_send..... | 35 |
| nx_telnet_server_packet_pool_set..... | 37 |
| nx_telnet_server_start..... | 38 |
| nx_telnet_server_stop | 39 |

Chapter 1

Introduction to Telnet

The Telnet Protocol (Telnet) is a protocol designed for transferring commands and responses between two nodes on the Internet. Telnet is a simple protocol that utilizes reliable Transmission Control Protocol (TCP) services to perform its transfer function. Because of this, Telnet is a highly reliable transfer protocol. Telnet is also one of the most used application protocols.

Telnet Requirements

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

The Telnet Server portion of the NetX Telnet package has one additional requirement. It requires complete access to TCP *well-known port 23* for handling all Client Telnet requests.

Telnet Constraints

The NetX Telnet protocol implements the Telnet standard. However, the interpretation and response of Telnet commands, indicated by a byte with the value of 255, is the responsibility of the application. The various Telnet commands and command parameters are defined in the *nx_telnet_client.h* and *nx_telnet_server.h* files.

Telnet Communication

As mentioned previously, the Telnet Server utilizes the *well-known TCP port 23* to field Client requests. Telnet Clients may use any available TCP port.

Telnet Authentication

Telnet authentication is the responsibility of the application's Telnet Server callback function. The application's Telnet Server "new connection" callback would typically prompt the Client for name and/or password. The Client would then be responsible for providing the information. The Server would then process the information in the "receive data" callback. This is where the application Server code would have to authenticate the information and decide whether or not it is valid.

Telnet New Connection Callback

The NetX Telnet Server calls the application specified callback function whenever a new Telnet Client request is received. The application specifies the callback function when the Telnet Server is created via the ***nx_telnet_server_create*** function. Typical actions of the "new connection" callback include sending a banner or prompt to the Client. This could very well include a prompt for login information.

The format of the application "new connection" callback routine is very simple and is defined below:

```
void telnet_new_connection(NX_Telnet_SERVER *server_ptr,
                          UINT logical_connection);
```

The input parameters are defined as follows:

| Parameter | Meaning |
|---------------------------|--|
| <i>server_ptr</i> | Pointer to the calling Telnet Server. |
| <i>logical_connection</i> | The internal logical connection for the Telnet Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_Telnet_MAX_CLIENTS-1. |

Telnet Receive Data Callback

The NetX Telnet Server calls the application specified callback function whenever a new Telnet Client data is received. The application specifies

the callback function when the Telnet Server is created via the ***nx_telnet_server_create*** function. Typical actions of the “new connection” callback include echoing the data back and/or parsing the data and providing data as a result of interpreting a command from the client.

Note that this callback routine must also release the supplied packet.

The format of the application “receive data” callback routine is very simple and is defined below:

```
void telnet_receive_data(NX_Telnet_SERVER *server_ptr,
                        UINT logical_connection, NX_PACKET *packet_ptr);
```

The input parameters are defined as follows:

| Parameter | Meaning |
|---------------------------|--|
| <i>server_ptr</i> | Pointer to the calling Telnet Server. |
| <i>logical_connection</i> | The internal logical connection for the Telnet Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_Telnet_MAX_CLIENTS-1. |
| <i>packet_ptr</i> | Pointer to packet containing the data from the Client. |

Telnet End Connection Callback

The NetX Telnet Server calls the application specified callback function whenever a Telnet Client ends the connection. The application specifies the callback function when the Telnet Server is created via the ***nx_telnet_server_create*** function. Typical actions of the “end connection” callback include cleaning up any Client specific data structures associated with the logical connection.

The format of the application “end connection” callback routine is very simple and is defined below:

```
void telnet_end_connection(NX_Telnet_SERVER *server_ptr,
                          UINT logical_connection);
```

The input parameters are defined as follows:

| Parameter | Meaning |
|---------------------------|--|
| <i>server_ptr</i> | Pointer to the calling Telnet Server. |
| <i>logical_connection</i> | The internal logical connection for the Telnet Server. This can be used by the application as an index into buffers and/or data structures specific for each Client connection. Its value ranges from 0 through NX_Telnet_MAX_CLIENTS-1. |

Telnet Option Negotiation

The NetX Telnet Server supports a limited set of Telnet options, `Echo` and `Suppress Go Ahead`.

To enable this feature the `NX_TELNET_SERVER_OPTION_DISABLE` must not be defined. By default it is not defined. The Telnet Server creates a packet pool in the *nx_telnet_server_create* service from which it allocates packets for sending telnet options requests to the Client. See “Configuration Options” for setting the packet payload (`NX_TELNET_SERVER_PACKET_PAYLOAD`) and packet pool size (`NX_TELNET_SERVER_PACKET_POOL_SIZE`) for this packet pool. It will delete this packet pool when the *nx_telnet_server_delete* service is called.

Upon making a connection with the Telnet Client, it will send out this set of telnet options to the Client if it has not received option requests from the Client:

```
will echo

dont echo

will sga
```

When it receives Telnet data from the Client, the Telnet Server checks if the first byte is the “IAC” code. If so, it will process all the options in the Client packet. Options not in the list above are ignored.

By default, the Telnet Server creates its own internal packet pool if `NX_TELNET_SERVER_OPTION_DISABLE` is not defined and it needs to transmit Telnet option commands. The Telnet Server packet pool is defined by

`NX_TELNET_SERVER_PACKET_PAYLOAD` and `NX_TELNET_SERVER_PACKET_POOLSIZE`. If, however, `NX_TELNET_SERVER_USER_CREATE_PACKET_POOL` is defined, the application must create the Telnet Server packet pool and set it as the Telnet Server packet pool by calling `_nx_telnet_server_packet_pool_set`. See Chapter 3 “Description of Telnet Services” for more details about this function.

Unlike the NetX Telnet Server, the NetX Telnet Client task thread does not automatically send and respond to received options from the Telnet Server. This must be done by the Telnet Client application.

Telnet Multi-Thread Support

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

Telnet RFCs

NetX Telnet is compliant with RFC854 and related RFCs.

Chapter 2

Installation and Use of Telnet

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

Product Distribution

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

| | |
|--|--|
| <code>nx_telnet_client.h</code> | Header file for Telnet Client for NetX |
| <code>nx_telnet_client.c</code> | C Source file for Telnet Client for NetX |
| <code>nx_telnet_server.h</code> | Header file for Telnet Server for NetX |
| <code>nx_telnet_server.c</code> | C Source file for Telnet Server for NetX |
| <code>nx_telnet.pdf</code> | PDF description of Telnet for NetX |
| <code>demo_netx_telnet.c</code> | NetX Telnet demonstration |

Telnet Installation

In order to use Telnet 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_telnet_client.h`, `nx_telnet_client.c`, `nx_telnet_server.c` and `nx_telnet_server.h` files should be copied into this directory.

Using Telnet

Using Telnet for NetX is easy. Basically, the application code must include the header files after it includes `tx_api.h` and `nx_api.h`, in order to use ThreadX and NetX. Once the header files are included, the application code is then able to make the Telnet function calls specified later in this guide. The application must also include `nx_telnet_client.c` and `nx_telnet_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 Telnet.

If no Telnet Client capabilities are required, the `nx_telnet_client.c` file may be omitted.

Note also that because Telnet utilizes NetX TCP services, TCP must be enabled with the *nx_tcp_enable* call prior to using Telnet.

Small Example System

An example of how easy it is to use NetX Telnet is described in Figure 1.1 that appears below. In this example, the Telnet include file *nx_telnet_client.h* and *nx_telnet_server.h* are brought in at line 7. Next, the Telnet Server is created in “*tx_application_define*” at line 112. Note that the Telnet Server control block “*Server*” was defined as a global variable at line 24 previously. After successful creation, an Telnet Server is started at line 121. At line 138 the Telnet Client is created. And finally, the Client sends a character at line 160 and reads the character back at line 182.

```

001 /* This is a small demo of Telnet on the high-performance NetX TCP/IP stack.
002    This demo relies on ThreadX and NetX to show a simple Telnet connection,
003    send, server echo, and then disconnection from the Telnet server. */
004
005 #include "tx_api.h"
006 #include "nx_api.h"
007 #include "nx_telnet_client.h"
008 #include "nx_telnet_server.h"
009 #define DEMO_STACK_SIZE 4096
010
011
012 /* Define the ThreadX and NetX object control blocks... */
013
014 TX_THREAD thread_0;
015 TX_THREAD thread_1;
016 NX_PACKET_POOL pool_0;
017 NX_PACKET_POOL pool_1;
018 NX_IP ip_0;
019 NX_IP ip_1;
020
021
022 /* Define Telnet objects. */
023
024 NX_Telnet_SERVER my_server;
025 NX_Telnet_CLIENT my_client;
026
027 /* Define the counters used in the demo application... */
028
029 ULONG error_counter;
030
031
032 /* Define function prototypes. */
033
034 void thread_0_entry(ULONG thread_input);
035 void _nx_ram_network_driver(struct NX_IP_DRIVER_STRUCT *driver_req);
036
037
038 /* Define the application's Telnet Server callback routines. */
039
040 void telnet_new_connection(NX_Telnet_SERVER *server_ptr,
041                            UINT logical_connection);
042 void telnet_receive_data(NX_Telnet_SERVER *server_ptr,
043                          UINT logical_connection, NX_PACKET *packet_ptr);
044 void telnet_connection_end(NX_Telnet_SERVER *server_ptr,
045                            UINT logical_connection);
046
047
048 /* Define main entry point. */
049
050 int main()
051 {

```

```

052
053     /* Enter the ThreadX kernel. */
054     tx_kernel_enter();
055 }
056
057
058 /* Define what the initial system looks like. */
059 void tx_application_define(void *first_unused_memory)
060 {
061
062     UINT    status;
063     CHAR    *pointer;
064
065
066     /* Setup the working pointer. */
067     pointer = (CHAR *) first_unused_memory;
068
069     /* Create the main thread. */
070     tx_thread_create(&thread_0, "thread 0", thread_0_entry, 0,
071                     pointer, DEMO_STACK_SIZE,
072                     2, 2, TX_NO_TIME_SLICE, TX_AUTO_START);
073     pointer = pointer + DEMO_STACK_SIZE;
074
075     /* Initialize the NetX system. */
076     nx_system_initialize();
077
078     /* Create packet pool. */
079     nx_packet_pool_create(&pool_0, "NetX Packet Pool 0",
080                           600, pointer, 8192);
081     pointer = pointer + 8192;
082
083     /* Create an IP instance. */
084     nx_ip_create(&ip_0, "NetX IP Instance 0", IP_ADDRESS(1, 2, 3, 4),
085                 0xFFFFFFFFUL, &pool_0, _nx_ram_network_driver,
086                 pointer, 4096, 1);
087     pointer = pointer + 4096;
088
089     /* Create another packet pool. */
090     nx_packet_pool_create(&pool_1, "NetX Packet Pool 1", 600, pointer, 8192);
091     pointer = pointer + 8192;
092
093     /* Create another IP instance. */
094     nx_ip_create(&ip_1, "NetX IP Instance 1", IP_ADDRESS(1, 2, 3, 5),
095                 0xFFFFFFFFUL, &pool_1, _nx_ram_network_driver,
096                 pointer, 4096, 1);
097     pointer = pointer + 4096;
098
099     /* Enable ARP and supply ARP cache memory for IP Instance 0. */
100     nx_arp_enable(&ip_0, (void *) pointer, 1024);
101     pointer = pointer + 1024;
102
103     /* Enable ARP and supply ARP cache memory for IP Instance 1. */
104     nx_arp_enable(&ip_1, (void *) pointer, 1024);
105     pointer = pointer + 1024;
106
107     /* Enable TCP processing for both IP instances. */
108     nx_tcp_enable(&ip_0);
109     nx_tcp_enable(&ip_1);
110
111     /* Create the NetX Telnet Server. */
112     status = nx_telnet_server_create(&my_server, "Telnet Server", &ip_0,
113                                     pointer, 2048, telnet_new_connection,
114                                     telnet_receive_data, telnet_connection_end);
115
116     /* Check for errors. */
117     if (status)
118         error_counter++;
119
120     /* Start the Telnet Server. */
121     status = nx_telnet_server_start(&my_server);
122
123     /* Check for errors. */
124     if (status)
125         error_counter++;
126 }
127
128
129 /* Define the test thread. */
130 void thread_0_entry(ULONG thread_input)
131 {
132

```

```

133 NX_PACKET    *my_packet;
134 UINT         status;
135
136
137     /* Create a TELNET client instance. */
138     status = nx_telnet_client_create(&my_client, "My Telnet Client",
139                                     &ip_1, 600);
140
141     /* Check status. */
142     if (status)
143         error_counter++;
144
145     /* Connect the Telnet client to the Telnet Server at port 23. */
146     status = nx_telnet_client_connect(&my_client, IP_ADDRESS(1,2,3,4), 23, 50);
147
148     /* Check status. */
149     if (status)
150         error_counter++;
151
152     /* Allocate a packet. */
153     status = nx_packet_allocate(&pool_0, &my_packet, NX_TCP_PACKET,
154                                NX_WAIT_FOREVER);
155
156     /* Build a simple 1-byte message. */
157     nx_packet_data_append(my_packet, "a", 1, &pool_0, NX_WAIT_FOREVER);
158
159     /* Send the packet to the Telnet Server. */
160     status = nx_telnet_client_packet_send(&my_client, my_packet, 50);
161
162     /* Check status. */
163     if (status)
164         error_counter++;
165
166     /* Pickup the Server header. */
167     status = nx_telnet_client_packet_receive(&my_client, &my_packet, 50);
168
169     /* Check status. */
170     if (status)
171         error_counter++;
172     else
173     {
174
175         /* At this point the packet should contain the Server's banner
176            message sent by the Server callback function below. Just
177            release it for this demo. */
178         nx_packet_release(my_packet);
179     }
180
181     /* Pickup the Server echo of the character. */
182     status = nx_telnet_client_packet_receive(&my_client, &my_packet, 50);
183
184     /* Check status. */
185     if (status)
186         error_counter++;
187     else
188     {
189
190         /* At this point the packet should contain the character 'a' that
191            we sent earlier. Just release the packet for now. */
192         nx_packet_release(my_packet);
193     }
194
195     /* Now disconnect form the Telnet Server. */
196     status = nx_telnet_client_disconnect(&my_client, 50);
197
198     /* Check status. */
199     if (status)
200         error_counter++;
201
202     /* Delete the Telnet Client. */
203     status = nx_telnet_client_delete(&my_client);
204
205     /* Check status. */
206     if (status)
207         error_counter++;
208 }
209
210
211 /* This routine is called by the NetX Telnet Server whenever a new Telnet client
212    connection is established. */
213 void telnet_new_connection(NX_Telnet_SERVER *server_ptr, UINT logical_connection)

```

```

214 {
215
216     UINT      status;
217     NX_PACKET  *packet_ptr;
218
219     /* Allocate a packet for client greeting. */
220     status = nx_packet_allocate(&pool_0, &packet_ptr, NX_TCP_PACKET, NX_NO_WAIT);
221
222     /* Determine if we have a packet. */
223     if (status == NX_SUCCESS)
224     {
225
226         /* Build a banner message and a prompt. */
227         nx_packet_data_append(packet_ptr,
228             "**** Welcome to NetX Telnet Server ****\r\n\r\n\r\n", 45,
229             &pool_0, NX_NO_WAIT);
230         nx_packet_data_append(packet_ptr, "NETX> ", 6, &pool_0, NX_NO_WAIT);
231
232         /* Send the packet to the client. */
233         status = nx_telnet_server_packet_send(server_ptr, logical_connection,
234             packet_ptr, 100);
235
236         if (status)
237             nx_packet_release(packet_ptr);
238     }
239 }
240
241
242 /* This routine is called by the NetX Telnet Server whenever data is
243 present on a Telnet client connection. */
244 void telnet_receive_data(NX_Telnet_SERVER *server_ptr, UINT logical_connection,
245     NX_PACKET *packet_ptr)
246 {
247
248     UINT      status;
249     UCHAR     alpha;
250
251
252     /* This demo just echoes the character back and on <cr,lf> sends a new prompt
253 back to the client. A real system would most likely buffer the character(s)
254 received in a buffer associated with the supplied logical connection and
255 process according to it. */
256
257     /* Just throw away carriage returns. */
258     if ((packet_ptr -> nx_packet_prepend_ptr[0] == '\r') &&
259         (packet_ptr -> nx_packet_length == 1))
260     {
261         nx_packet_release(packet_ptr);
262         return;
263     }
264
265     /* Setup new line on line feed. */
266     if ((packet_ptr -> nx_packet_prepend_ptr[0] == '\n') ||
267         (packet_ptr -> nx_packet_prepend_ptr[1] == '\n'))
268     {
269
270         /* Clean up the packet. */
271         packet_ptr -> nx_packet_length = 0;
272         packet_ptr -> nx_packet_prepend_ptr =
273             packet_ptr -> nx_packet_data_start + NX_TCP_PACKET;
274         packet_ptr -> nx_packet_append_ptr =
275             packet_ptr -> nx_packet_data_start + NX_TCP_PACKET;
276
277         /* Build the next prompt. */
278         nx_packet_data_append(packet_ptr, "\r\nNETX> ", 8, &pool_0, NX_NO_WAIT);
279
280         /* Send the packet to the client. */
281         status = nx_telnet_server_packet_send(server_ptr, logical_connection,
282             packet_ptr, 100);
283
284         if (status)
285             nx_packet_release(packet_ptr);
286         return;
287     }
288
289     /* Pickup first character (usually only one from client). */
290     alpha = packet_ptr -> nx_packet_prepend_ptr[0];
291
292     /* Echo character. */
293     status = nx_telnet_server_packet_send(server_ptr, logical_connection,
294         packet_ptr, 100);

```



```

295
296     if (status)
297         nx_packet_release(packet_ptr);
298
299     /* Check for a disconnection. */
300     if (alpha == 'q')
301     {
302
303         /* Initiate server disconnection. */
304         nx_telnet_server_disconnect(server_ptr, logical_connection);
305     }
306 }
307
308
309 /* This routine is called by the NetX Telnet Server whenever the
310    client disconnects. */
311 void telnet_connection_end(NX_Telnet_SERVER *server_ptr,
312                           UINT logical_connection)
313 {
314
315     /* Cleanup any application specific connection or buffer information. */
316 }

```

Figure 1.1 Example of Telnet use with NetX

Configuration Options

There are several configuration options for building Telnet for NetX. These #defines can be set by the application prior to inclusion of *nx_telnet_server.h* and *nx_telnet_client.h*

Following is a list of all options, where each is described in detail:

| Define | Meaning |
|-------------------------------------|--|
| NX_DISABLE_ERROR_CHECKING | Defined, this option removes the basic Telnet error checking. It is typically used after the application has been debugged. |
| NX_TELNET_MAX_CLIENTS | The maximum number of Telnet Clients supported by the Server thread. By default, this value is defined as 4 to specify a maximum of 4 clients at a time. |
| NX_TELNET_SERVER_PRIORITY | The priority of the Telnet Server thread. By default, this value is defined as 16 to specify priority 16. |
| NX_TELNET_TOS | Type of service required for the Telnet TCP requests. By default, this value is defined as NX_IP_NORMAL to indicate normal IP packet service. |
| NX_TELNET_FRAGMENT_OPTION | Fragment enable for Telnet TCP requests. By default, this value is NX_DONT_FRAGMENT to disable Telnet TCP fragmenting. |
| NX_TELNET_SERVER_WINDOW_SIZE | Server socket window size. By default, this value is 2048 bytes. |
| NX_TELNET_TIME_TO_LIVE | Specifies the number of routers this packet can pass before it |

is discarded. The default value is set to 0x80.

NX_TELNET_SERVER_TIMEOUT

Specifies the number of ThreadX ticks that internal services will suspend for. The default value is set to 10 seconds.

NX_TELNET_ACTIVITY_TIMEOUT

Specifies the number of seconds that can elapse without any activity before the Server disconnects the Client connection. The default value is set to 600 seconds.

NX_TELNET_TIMEOUT_PERIOD

Specifies the number of seconds between checking for Client activity timeouts. The default value is set to 60 seconds.

NX_TELNET_SERVER_OPTION_DISABLE

Defined, Telnet option negotiation is disabled. By default this option is not defined.

NX_TELNET_SERVER_USER_CREATE_PACKET_POOL

If defined, the Telnet Server packet pool must be created externally. This is only meaningful if `NX_TELNET_SERVER_OPTION_DISABLE` is not defined. By default this option is not defined and the Telnet Server thread creates its own packet pool.

NX_TELNET_SERVER_PACKET_PAYLOAD

Defines the size of the packet payload created by the Telnet Server for option negotiation. Note that the Telnet Server only creates this packet pool if `NX_TELNET_SERVER_OPTION_DISABLE` is not defined (Telnet options are

enabled). The default value of this option is 300.

NX_TELNET_SERVER_PACKET_POOL_SIZE

Defines the size of the Telnet Server packet pool used for Telnet negotiations. Note that the Telnet Server only creates this packet pool if NX_TELNET_SERVER_OPTION_DISABLE is not defined (Telnet options are enabled). The default value of this option is 2048 (~5-6 packets).

Chapter 3

Description of Telnet Services

This chapter contains a description of all NetX Telnet 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_telnet_client_connect`
Connect a Telnet Client

`nx_telnet_client_create`
Create a Telnet Client

`nx_telnet_client_delete`
Delete a Telnet Client

`nx_telnet_client_disconnect`
Disconnect a Telnet Client

`nx_telnet_client_packet_receive`
Receive packet via Telnet Client

`nx_telnet_client_packet_send`
Send packet via Telnet Client

`nx_telnet_server_create`
Create a Telnet Server

`nx_telnet_server_delete`
Delete a Telnet Server

`nx_telnet_server_disconnect`
Disconnect a Telnet Client

`nx_telnet_server_packet_send`
Send packet through Client connection

`nx_telnet_server_packet_pool_set`
Set packet pool as Telnet Server packet pool

`nx_telnet_server_start`
Start a Telnet Server

`nx_telnet_server_stop`
Stop a Telnet Server

| status | | Actual NetX completion status |
|---------------------|--------|---------------------------------|
| NX_PTR_ERROR | (0x16) | Invalid Client pointer. |
| NX_IP_ADDRESS_ERROR | (0x21) | Invalid IP address. |
| NX_CALLER_ERROR | (0x11) | Invalid caller of this service. |

Allowed From

Threads

Example

```

/* Connect the Telnet Client instance "my_client" to the Server at
   IP address 1.2.3.4 and port 23. */
status = nx_telnet_client_connect(&my_client, IP_ADDRESS(1,2,3,4), 23, 100);

/* If status is NX_SUCCESS the Telnet Client instance was successfully
   connected to the Telnet Server. */

```


nx_telnet_client_create

Create a Telnet Client

Prototype

```
UINT nx_telnet_client_create(NX_Telnet_CLIENT *client_ptr,
                             CHAR *client_name, NX_IP *ip_ptr, ULONG window_size);
```

Description

This service creates a Telnet Client instance.

Input Parameters

| | |
|--------------------|---|
| client_ptr | Pointer to Telnet Client control block. |
| client_name | Name of Client instance. |
| ip_ptr | Pointer to IP instance. |
| window_size | Size of TCP receive window for this Client. |

Return Values

| | | |
|---------------------|--------|-------------------------------|
| NX_SUCCESS | (0x00) | Successful Client create. |
| status | | Actual NetX completion status |
| NX_PTR_ERROR | (0x16) | Invalid Client or IP pointer. |

Allowed From

Initialization, Threads

Example

```
/* Create the Telnet Client instance "my_client" on the IP instance "ip_0". */
status = nx_telnet_client_create(&my_client, "My Telnet Client", &ip_0, 2048);

/* If status is NX_SUCCESS the Telnet Client instance was successfully
   created. */
```

nx_telnet_client_delete

Delete a Telnet Client

Prototype

```
UINT nx_telnet_client_delete(NX_Telnet_CLIENT *client_ptr);
```

Description

This service deletes a previously created Telnet Client instance.

Input Parameters

client_ptr Pointer to Telnet Client control block.

Return Values

| | | |
|-----------------------------------|--------|---------------------------------|
| NX_SUCCESS | (0x00) | Successful Client delete. |
| NX_TELNET_NOT_DISCONNECTED | (0xF4) | Client still connected. |
| NX_PTR_ERROR | (0x16) | Invalid Client pointer. |
| NX_CALLER_ERROR | (0x11) | Invalid caller of this service. |

Allowed From

Threads

Example

```
/* Delete the Telnet Client instance "my_client". */
status = nx_telnet_client_delete(&my_client);

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

nx_telnet_client_disconnect

Disconnect a Telnet Client

Prototype

```
UINT nx_telnet_client_disconnect(NX_Telnet_CLIENT *client_ptr,
                                ULONG wait_option);
```

Description

This service disconnects a previously connected Telnet Client instance.

Input Parameters

| | |
|--------------------|---|
| client_ptr | Pointer to Telnet Client control block. |
| wait_option | Defines how long the service will wait for the Telnet Client disconnect. The wait options are defined as follows: timeout value (0x00000001 through 0xFFFFFFFFE) TX_WAIT_FOREVER (0xFFFFFFFFF) Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the Telnet Server responds to the request. Selecting a numeric value (1-0xFFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for the Telnet Server response. |

Return Values

| | | |
|--------------------------------|--------|---------------------------------|
| NX_SUCCESS | (0x00) | Successful Client disconnect. |
| NX_TELNET_NOT_CONNECTED | (0xF3) | Client not connected. |
| NX_PTR_ERROR | (0x16) | Invalid Client pointer. |
| NX_CALLER_ERROR | (0x11) | Invalid caller of this service. |

Allowed From

Threads

Example

```
/* Disconnect the Telnet Client instance "my_client". */  
status = nx_telnet_client_disconnect(&my_client, 100);  
  
/* If status is NX_SUCCESS the Telnet Client instance was successfully  
   disconnected. */
```

nx_telnet_client_packet_receive

Receive packet via Telnet Client

Prototype

```
UINT nx_telnet_client_packet_receive(NX_Telnet_CLIENT *client_ptr,
                                     NX_PACKET **packet_ptr, ULONG wait_option);
```

Description

This service receives a packet from the previously connected Telnet Client instance.

Input Parameters

| | |
|--------------------|---|
| client_ptr | Pointer to Telnet Client control block. |
| packet_ptr | Pointer to the destination for the received packet. |
| wait_option | Defines how long the service will wait for the Telnet Client packet receive. 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 Telnet 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 Telnet Server response.

Return Values

| | | |
|---------------------|--------|-----------------------------------|
| NX_SUCCESS | (0x00) | Successful Client packet receive. |
| status | | Actual NetX completion status |
| NX_PTR_ERROR | (0x16) | Invalid Client or packet pointer. |

| | | |
|-----------------|--------|---------------------------------|
| NX_CALLER_ERROR | (0x11) | Invalid caller of this service. |
|-----------------|--------|---------------------------------|

Allowed From

Threads

Example

```
/* Receive a packet from the Telnet Client instance "my_client". */
status = nx_telnet_client_packet_receive(&my_client, &my_packet, 100);

/* If status is NX_SUCCESS the "my_packet" pointer contains data received from
   the Telnet Client connection. */
```

nx_telnet_client_packet_send

Send packet via Telnet Client

Prototype

```
UINT nx_telnet_client_packet_send(NX_Telnet_CLIENT *client_ptr,
                                  NX_PACKET *packet_ptr, ULONG wait_option);
```

Description

This service sends a packet through the previously connected Telnet Client instance.

Input Parameters

| | |
|--------------------|--|
| client_ptr | Pointer to Telnet Client control block. |
| packet_ptr | Pointer to the packet to send. |
| wait_option | Defines how long the service will wait for the Telnet Client packet send. 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 Telnet 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 Telnet Server response.

Return Values

| | | |
|------------------------|--------|---------------------------------|
| NX_SUCCESS | (0x00) | Successful packet send. |
| NX_PTR_ERROR | (0x16) | Invalid pointer input |
| NX_CALLER_ERROR | (0x11) | Invalid caller of this service. |

Allowed From

Threads

Example

```
/* Send a packet via the Telnet Client instance "my_client". */  
status = nx_telnet_client_packet_send(&my_client, my_packet, 100);  
  
/* If status is NX_SUCCESS the packet was successfully sent. */
```


nx_telnet_server_create

Create a Telnet Server

Prototype

```
UINT nx_telnet_server_create(NX_Telnet_SERVER *server_ptr,
                             CHAR *server_name, NX_IP *ip_ptr,
                             VOID *stack_ptr, ULONG stack_size,
                             void (*new_connection)(struct NX_Telnet_SERVER_STRUCT
                                                      *telnet_server_ptr, UINT logical_connection),
                             void (*receive_data)(struct NX_Telnet_SERVER_STRUCT
                                                    *telnet_server_ptr, UINT logical_connection,
                                                    NX_PACKET *packet_ptr),
                             void (*connection_end)(struct NX_Telnet_SERVER_STRUCT
                                                      *telnet_server_ptr, UINT logical_connection));
```

Description

This service creates a Telnet Server instance on the specified IP instance.

Input Parameters

| | |
|-----------------------|---|
| server_ptr | Pointer to Telnet Server control block. |
| server_name | Name of Telnet Server instance. |
| ip_ptr | Pointer to associated IP instance. |
| stack_ptr | Pointer to stack for the internal Server thread. |
| sack_size | Size of the stack, in bytes. |
| new_connection | Application callback routine function pointer. This routine is called whenever a new Telnet Client connection request is detected by the Server. |
| receive_data | Application callback routine function pointer. This routine is called whenever a new Telnet Client data is present on the connection. This routine is responsible for releasing the packet. |
| end_connection | Application callback routine function pointer. This routine is called whenever a Telnet Client connection is disconnected by the Client. The Server can also disconnect via the <i>nx_telnet_server_disconnect</i> service described below. |

Return Values

| | | |
|------------------------|--------|---------------------------------|
| NX_SUCCESS | (0x00) | Successful Server delete. |
| NX_PTR_ERROR | (0x07) | Invalid Server pointer. |
| NX_CALLER_ERROR | (0x11) | Invalid caller of this service. |

Allowed From

Initialization, Threads

Example

```
/* Create a Telnet Server instance "my_server". */
status = nx_telnet_server_create(&my_server, "Telnet Server", &ip_0,
                                pointer, 2048, telnet_new_connection, telnet_receive_data,
                                telnet_connection_end);

/* If status is NX_SUCCESS the Telnet Server was successfully created. */
```

nx_telnet_server_delete

Delete a Telnet Server

Prototype

```
UINT nx_telnet_server_delete(NX_Telnet_SERVER *server_ptr);
```

Description

This service deletes a previously created Telnet Server instance.

Input Parameters

server_ptr Pointer to Telnet Server control block.

Return Values

| | | |
|------------------------|--------|---------------------------------|
| NX_SUCCESS | (0x00) | Successful Server delete. |
| NX_PTR_ERROR | (0x16) | Invalid Server pointer. |
| NX_CALLER_ERROR | (0x11) | Invalid caller of this service. |

Allowed From

Threads

Example

```
/* Delete the Telnet Server instance "my_server". */
status = nx_telnet_server_delete(&my_server);

/* If status is NX_SUCCESS the Telnet Server was successfully deleted. */
```

nx_telnet_server_disconnect

Disconnect a Telnet Client

Prototype

```
UINT nx_telnet_server_disconnect(NX_Telnet_SERVER *server_ptr,
                                UINT logical_connection);
```

Description

This service disconnects a previously connected Client on this Telnet Server instance. This routine is typically called from the application's receive data callback function in response to a condition detected in the data received.

Input Parameters

server_ptr Pointer to Telnet Server control block.

logical_connection Logical connection corresponding the Client connection on this Server. Valid value range from 0 through NX_TELENET_MAX_CLIENTS.

Return Values

| | | |
|------------------------|--------|---------------------------------|
| NX_SUCCESS | (0x00) | Successful Server disconnect. |
| NX_OPTION_ERROR | (0x0A) | Invalid logical connection. |
| NX_PTR_ERROR | (0x16) | Invalid Server pointer. |
| NX_CALLER_ERROR | (0x11) | Invalid caller of this service. |

Allowed From

Threads

Example

```
/* Disconnect the Telnet Client associated with logical connection 2 on
   the Telnet Server instance "my_server". */
status = nx_telnet_server_disconnect(&my_server, 2);

/* If status is NX_SUCCESS the Client on logical connection 2 was
   disconnected. */
```

nx_telnet_server_get_open_connection_count

Return number of currently open connections

Prototype

```
UINT nx_telnet_server_get_open_connection_count(NX_TELNET_SERVER
                                                *server_ptr, UINT *connection_count);
```

Description

This service returns the number of currently connected Telnet Clients.

Input Parameters

server_ptr Pointer to Telnet Server control block.

Connection_count
 Pointer to memory to store connection count

Return Values

| | | |
|------------------------|--------|---------------------------------|
| NX_SUCCESS | (0x00) | Successful completion. |
| NX_PTR_ERROR | (0x07) | Invalid Server pointer. |
| NX_CALLER_ERROR | (0x11) | Invalid caller of this service. |

Allowed From

Threads

Example

```
/* Get the number of Telnet Clients connected to the Server. */
status = nx_telnet_server_get_open_connection_count(&my_server, &conn_count);
/* If status is NX_SUCCESS the conn_count holds the number of open connections.
*/
```

nx_telnet_server_packet_send

Send packet through Client connection

Prototype

```
UINT nx_telnet_server_packet_send(NX_Telnet_SERVER *server_ptr,
                                  UINT logical_connection, NX_PACKET *packet_ptr,
                                  ULONG wait_option);
```

Description

This service sends a packet to the Client connection on this Telnet Server instance. This routine is typically called from the application's receive data callback function in response to a condition detected in the data received.

Input Parameters

| | |
|---------------------------|--|
| server_ptr | Pointer to Telnet Server control block. |
| logical_connection | Logical connection corresponding the Client connection on this Server. Valid value range from 0 through NX_TELENET_MAX_CLIENTS. |
| packet_ptr | Pointer to the received packet. |
| wait_option | <p>Defines how long the service will wait for the Telnet Server packet send. The wait options are defined as follows:</p> <p>timeout value (0x00000001 through 0xFFFFFFFF)</p> <p>TX_WAIT_FOREVER (0xFFFFFFFF)</p> <p>Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until the Telnet Server responds to the request.</p> <p>Selecting a numeric value (1-0xFFFFFFFF) specifies the maximum number of timer-ticks to stay suspended while waiting for the Telnet Server response.</p> |

Return Values

| | | |
|-------------------------|--------|---------------------------------|
| NX_SUCCESS | (0x00) | Successful Server packet send. |
| NX_TELNET_FAILED | (0xF2) | Server socket send failed. |
| NX_OPTION_ERROR | (0x0A) | Invalid logical connection. |
| NX_PTR_ERROR | (0x16) | Invalid Server pointer. |
| NX_CALLER_ERROR | (0x11) | Invalid caller of this service. |

Allowed From

Threads

Example

```
/* Send a packet to the Telnet Client associated with logical connection 2 on
   the Telnet Server instance "my_server". */
status = nx_telnet_server_packet_send(&my_server, 2, my_packet, 100);

/* If status is NX_SUCCESS the packet was sent to the Client on logical
   connection 2. */
```

nx_telnet_server_packet_pool_set

Set previously created packet pool as Telnet Server pool

Prototype

```
UINT nx_telnet_server_packet_pool_set(NX_TELNET_SERVER *server_ptr,
                                      NX_PACKET_POOL *packet_pool_ptr);
```

Description

This service sets a previously created packet pool as the Telnet Server packet pool if `NX_TELNET_SERVER_USER_CREATE_PACKET_POOL` is defined. It also requires that `NX_TELNET_SERVER_OPTION_DISABLE` not be defined such that the Telnet Server needs a packet pool to transmit Telnet options to Telnet clients.

This permits applications to create the packet pool in different memory e.g. no cache memory, than the Telnet Server stack. Note that if this function does not check if the Telnet Server packet pool is already set. If it is called on a non NULL Telnet Server packet pool pointer, it will overwrite it and replace the existing packet pool with packet pool pointed to by the input pointer.

Input Parameters

| | |
|------------------------|---|
| server_ptr | Pointer to Telnet Server control block |
| packet_pool_ptr | Pointer to previously created packet pool |

Return Values

| | | |
|---------------------|--------|--------------------------------|
| NX_SUCCESS | (0x00) | Successful Server packet send. |
| NX_PTR_ERROR | (0x07) | Invalid Server pointer. |

Allowed From

Init, Threads

Example

```
status = nx_packet_pool_create(&telnet_server_packet_pool, "Telnet Server Packet
                             Pool", telnet_server_pool_area, 600*10);

/* Set the packet pool as the Telnet Server packet pool. */
status = nx_telnet_server_packet_pool_set(&my_server,
                                          &telnet_server_packet_pool);

/* If status is NX_SUCCESS the packet pool is set as Telnet Server pool. */
```


`nx_telnet_server_start`

Start a Telnet Server

Prototype

```
UINT nx_telnet_server_start(NX_Telnet_SERVER *server_ptr);
```

Description

This service starts a previously created Telnet Server instance.

Input Parameters

| | |
|-------------------|---|
| server_ptr | Pointer to Telnet Server control block. |
|-------------------|---|

Return Values

| | | |
|---------------------------------|--------|--------------------------|
| NX_SUCCESS | (0x00) | Successful Server start. |
| NX_TELNET_NO_PACKET_POOL | (0xF6) | No packet pool set |
| NX_PTR_ERROR | (0x16) | Invalid Server pointer. |

Allowed From

Initialization, Threads

Example

```
/* Start the Telnet Server instance "my_server". */
status = nx_telnet_server_start(&my_server);

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

nx_telnet_server_stop

Stop a Telnet Server

Prototype

```
UINT nx_telnet_server_stop(NX_Telnet_SERVER *server_ptr);
```

Description

This service stops a previously created and started Telnet Server instance.

Input Parameters

server_ptr Pointer to Telnet Server control block.

Return Values

| | | |
|------------------------|--------|---------------------------|
| NX_SUCCESS | (0x00) | Successfully stopped |
| NX_PTR_ERROR | (0x16) | Invalid Server pointer. |
| NX_CALLER_ERROR | (0x11) | Invalid caller of service |

Allowed From

Threads

Example

```
/* Stop the Telnet Server instance "my_server". */
status = nx_telnet_server_stop(&my_server);

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



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 Telnet Protocol (Telnet) User Guide

Publication Date: Rev.5.10 Nov 7, 2018

Published by: Renesas Electronics Corporation

NetX Telnet Protocol (Telnet) User Guide