

# NetX™

## Simple Network Time Protocol (SNTP) Client

### 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 SNTP Client for the Renesas Synergy platform, please use the following information.

## **Broadcast Support**

**Page 4:** Broadcast support has not been tested for SSP v1.5.0.

## **Multiple Network Interface**

**Page 8:** Multiple network interface support has not been tested for SSP v1.5.0.

## **Installation**

**Page 9:** If you are using Renesas Synergy SSP and the e<sup>2</sup> studio ISDE, SNTP Client will already be installed. You can ignore the NetX SNTP Client Installation section.



## **NetX Simple Network Time Protocol (SNTP) Client User Guide**

**Express Logic, Inc.**

858.613.6640  
Toll Free 888.THREADX  
FAX 858.521.4259

[www.expresslogic.com](http://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 SNTP .....	4
NetX SNTP Client Requirements .....	4
NetX SNTP Client Limitations .....	4
NetX SNTP Client Operation .....	5
Multiple Network Interfaces .....	8
SNTP and NTP RFCs .....	8
Chapter 2 Installation and Use of NetX SNTP Client .....	9
Product Distribution .....	9
NetX SNTP Client Installation .....	9
Using NetX SNTP Client .....	9
Small Example System .....	10
Configuration Options .....	18
Chapter 3 Description of NetX SNTP Client Services .....	24
nx_sntp_client_create .....	26
nx_sntp_client_delete .....	28
nx_sntp_client_get_local_time .....	29
nx_sntp_client_initialize_broadcast .....	30
nx_sntp_client_initialize_unicast .....	32
nx_sntp_client_receiving_updates .....	33
nx_sntp_client_request_unicast_time .....	34
nx_sntp_client_run_broadcast .....	35
nx_sntp_client_run_unicast .....	36
nx_sntp_client_set_local_time .....	39
nx_sntp_client_set_time_update_notify .....	40
nx_sntp_client_stop .....	41
nx_sntp_client_utility_display_date_time .....	42
nx_sntp_client_utility_msecs_to_fraction .....	44
Appendix A: SNTP Fatal Error Codes .....	45

# Chapter 1

## Introduction to SNTP

The Simple Network Time Protocol (SNTP) is a protocol designed for synchronizing clocks over the Internet. SNTP Version 4 is a simplified protocol based on the Network Time Protocol (NTP). It utilizes User Datagram Protocol (UDP) services to perform time updates in a simple, stateless protocol. Though not as complex as NTP, SNTP is highly reliable and accurate. In most places of the Internet of today, SNTP provides accuracies of 1-50 milliseconds, depending on the characteristics of the synchronization source and network paths. SNTP has many options to provide reliability of receiving time updates. Ability to switch to alternative servers, applying back off polling algorithms and automatic time server discovery are just a few of the means for an SNTP client to handle a variable Internet time service environment. What it lacks in precision it makes up for in simplicity and ease of implementation. SNTP is intended primarily for providing comprehensive mechanisms to access national time and frequency dissemination (e.g. NTP server) services.

## NetX SNTP Client Requirements

The NetX SNTP Client requires that an IP instance has already been created. In addition, UDP must be enabled on that same IP instance and should have access to the *well-known* port 123 for sending time data to an SNTP Server, although alternative ports will work as well. Broadcast clients should bind the UDP port their broadcast server is sending on, usually 123. The NetX SNTP Client application must have one or more IP SNTP Server addresses.

## NetX SNTP Client Limitations

Precision in local time representation in NTP time updates handled by the SNTP Client API is limited to millisecond resolution.

The SNTP Client only holds a single SNTP Server address at any time. If that Server appears to be no longer valid, the application must stop the SNTP Client task, and reinitialize it with another SNTP server address, using either broadcast or unicast SNTP communication.

The SNTP Client does not support multicast.

NetX SNTP Client does not support authentication mechanisms for verifying received packet data.

## NetX SNTP Client Operation

RFC 4330 recommends that SNTP clients should operate only at the highest stratum of their local network and preferably in configurations where no NTP or SNTP client is dependent them for synchronization. Stratum level reflects the host position in the NTP time hierarchy where stratum 1 is the highest level (a root time server) and 15 is the lowest allowed level (e.g. Client). The SNTP Client default minimum stratum is 2.

The NetX SNTP Client can operate in one of two basic modes, unicast or broadcast, to obtain time over the Internet. In unicast mode, the Client polls its SNTP Server on regular intervals and waits to receive a reply from that Server. When one is received, the Client verifies that the reply contains a valid time update by applying a set of 'sanity checks' recommended by RFC 4330. The Client then applies the time difference, if any, with the Server clock to its local clock. In broadcast mode, the Client merely listens for time update broadcasts and maintains its local clock after applying a similar set of sanity checks to verify the update time data. Sanity checks are described in detail in the **SNTP Sanity Checks** section below.

Before the Client can run in either mode, it must establish its operating parameters. This is done by calling either *nx\_sntp\_client\_initialize\_unicast* or *nx\_sntp\_client\_initialize\_broadcast* for unicast or broadcast modes, respectively. These serves set the time outs for maximum time lapse without a valid update, the limit on consecutive invalid updates received, a polling interval for unicast mode, operation mode e.g. unicast vs. broadcast, and SNTP Server.

If the maximum time lapse or maximum invalid updates received is exceeded, the SNTP Client continues to run but sets the current SNTP Server status to invalid. The application can poll the SNTP Client using the *nx\_sntp\_client\_receiving\_updates* service to verify the SNTP Server is still sending valid updates. If not, it should stop the SNTP Client thread using the *nx\_sntp\_client\_stop* service and call either of the two initialize services to set another SNTP Server address. To restart the SNTP Client, the application calls *nx\_sntp\_client\_run\_broadcast* or *nx\_sntp\_client\_run\_unicast*. Note that the application can change SNTP



Client operating mode in the initialize call to switch to unicast or broadcast as desired.

### Local Clock Operation

The SNTP time based on the number of seconds on the master NTP clock, or number of seconds elapsed in the first NTP epoch e.g. from Jan 1 **1900 00:00:00** to Jan 1 **1999 00:00:00**. The significance of 01-01-1999 was when the last leap second occurred. This value is defined as follows:

```
#define NTP_SECONDS_AT_01011999          0xBA368E80
```

Before the SNTP Client runs, the application can optionally initialize the SNTP Client local time for the Client to use as a baseline time. To do so, it must use the *nx\_sntp\_client\_set\_local\_time* service. This takes the time in NTP format, seconds and fraction, where fraction is the milliseconds in the NTP condensed time. Ideally the application can obtain an SNTP time from an independent source. There is no API for converting year, month, date and time to an NTP time in the NetX SNTP Client. For a description of NTP time format, refer to *RFC4330 "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI"*.

If no base local time is supplied when the SNTP Client starts up, the SNTP Client will accept the SNTP updates without comparing to its local time on the first update. Thereafter it will apply the maximum and minimum time update values to determine if it will modify its local time.

To obtain the SNTP Client local time, the application can use the *nx\_sntp\_client\_get\_local\_time* service.

### SNTP Sanity Checks

The Client examines the incoming packet for the following criteria:

- Source IP address must match the current server IP address.
- Sender source port must match with the current server source port.
- Packet length must be the minimum length to hold an SNTP time message.

Next, the time data is extracted from the packet buffer to which the Client then applies a set of specific 'sanity checks':

- The Leap Indicator set to 3 indicates the Server is not synchronized. The Client should attempt to find an alternative server.
- A stratum field set to zero is known as a Kiss of Death (KOD) packet. The SMTP Client KOD handler for this situation is a user defined callback. The small example demo file contains a simple KOD handler for this situation. The Reference ID field optionally contains a code indicating the reason for the KOD reply. At any rate, the KOD handler must indicate how to handle receiving a kiss of death from the SNTP Server. Typically it will want to reinitialize the SNTP Client with another SNTP Server.
- The Server SNTP version, stratum and mode of operation must be matched to the Client service.
- If the Client is configured with a server clock dispersion maximum, the Client checks the server clock dispersion on the first update received only, and if it exceeds the Client maximum, the Client rejects the Server.
- The Server time stamp fields must also pass specific checks. For the unicast Server, all time fields must be filled in and cannot be NULL. The Origination time stamp must equal the Transmit time stamp in the Client's SNTP time message request. This protects the Client from malicious intruders and rogue Server behavior. The broadcast Server need only fill in the Transmit time stamp. Since it does not receive anything from the Client it has no Receive or Origination fields to fill in.

A failed sanity check brands a time update as an invalid time update. The SNTP Client sanity check service tracks the number of consecutive invalid time updates received from the same Server.

If *nx\_sntp\_client\_apply\_sanity\_check* returns a unsuccessful status to the SNTP Client, the SNTP Client increments the invalid time update count.

If the Server time update passes the sanity checks, the Client then attempts to process the time data to its local time. If the Client is configured for round trip calculation, e.g. the time from sending an

update request to the time one is received, the round trip time is calculated. This value is halved and then added to the Server's time.

Next, if this is the first update received from the current SNTP Server, the SNTP Client determines if it should ignore the difference between the Server and Client local time. Thereafter all updates from the SNTP Server are evaluated for the difference with the Client local time. The difference between Client and Server time is compared with `NX_SNTP_CLIENT_MAX_TIME_ADJUSTMENT`. If it exceeds this value, the data is thrown out. If the difference is less than the `NX_SNTP_CLIENT_MIN_TIME_ADJUSTMENT` the difference is considered too small to require adjustment.

Passing all these checks, the time update is then applied to the SNTP Client with some corrections for delays in internal SNTP Client processing.

## Multiple Network Interfaces

NetX SNTP Client supports devices with multiple network interfaces.

## SNTP and NTP RFCs

NetX SNTP client is compliant with RFC4330 "Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI" and related RFCs.

## Chapter 2

# Installation and Use of NetX SNTP Client

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

## Product Distribution

SNTP 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:

<b><code>nx_sntp_client.c</code></b>	SNTP Client C source file
<b><code>nx_sntp_client.h</code></b>	SNTP Client Header file
<b><code>demo_netx_sntp_client.c</code></b>	Demonstration SNTP Client application
<b><code>nx_sntp.docx</code></b>	NetX SNTP Client User Guide

## NetX SNTP Client Installation

In order to use SNTP 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 NetX SNTP Client files *nx\_sntp\_client.c* and *nx\_sntp\_client.h* (*nx\_sntp\_client.c* and *nx\_sntp\_client.h* in NetX) should be copied into this directory.

## Using NetX SNTP Client

Using NetX SNTP Client is easy. Basically, the application code must include *nx\_sntp\_client.h* after it includes *tx\_api.h*, *fx\_api.h*, and *nx\_api.h*, in order to use ThreadX and NetX, respectively. Once *nx\_sntp\_client.h* is included, the application code is then able to make the SNTP function calls specified later in this guide. The application must also include *nx\_sntp\_client.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 SNTP Client.

Note that since the NetX SNTP Client utilizes NetX UDP services, UDP must be enabled with the *nx\_udp\_enable* call prior to using SNTP services.

## Small Example System

An example of how to use NetX SNTP is shown below. In this example, the SNTP include file *nx\_sntp\_client.h* is brought in at line 12. The SNTP Client is created in “*tx\_application\_define*” on line 148. Note that the kiss of death and leap second handler functions are optional when creating the SNTP Client.

Then the SNTP Client must be initialized for either unicast or broadcast mode.

SNTP Client initially writes Server time updates to its own internal data structure. This is not the same as the device local time. The device local time can be set as a baseline time in the SNTP Client before starting the SNTP Client thread. This is useful if the SNTP Client is configured (NX\_Sntp\_CLIENT\_IGNORE\_MAX\_ADJUST\_STARTUP set to NX\_FALSE) to compare the first Server update to the NX\_Sntp\_CLIENT\_MAX\_ADJUSTMENT (default value 180 milliseconds). Otherwise the SNTP Client will set the initial local time directly when it gets the first update from the Server.

A baseline time is applied to the SNTP Client on line 224 using the *nx\_sntp\_client\_set\_local\_time* service.

The SNTP Client is started on started at line 236 and 238 for unicast and broadcast mode respectively. The application then periodically checks for updates. The *nx\_sntp\_client\_receiving\_updates* service verifies that the SNTP Client is currently receiving valid updates. If so, it will retrieve the latest update time using the *nx\_sntp\_client\_get\_local\_time* service on line 261.

The SNTP Client can be stopped at any time using the *nx\_sntp\_client\_stop* service (282) if for example it detects the SNTP Client is no longer receiving valid updates.. To restart the Client, the application must call either the unicast or broadcast initialize service and then call either unicast or broadcast run services. Note that the SNTP Client can switch SNTP servers and modes (unicast or broadcast) while stopped.

```

1  /*
2  This is a small demo of the NetX SNTP Client on the high-performance NetX TCP/IP stack.
3  This demo relies on Thread, NetX and NetX SNTP Client API to execute the Simple Network Time

```

```

4      Protocol in unicast and broadcast modes.
5
6      */
7
8
9      #include <stdio.h>
10     #include "nx_api.h"
11     #include "nx_ip.h"
12     #include "nx_sntp_client.h"
13
14     /* Set up generic network driver for demo program. */
15     void _nx_ram_network_driver(struct NX_IP_DRIVER_STRUCT *driver_req);
16
17     /* Application defined services of the NetX SNTP Client. */
18
19     UINT leap_second_handler(NX_SNTP_CLIENT *client_ptr, UINT leap_indicator);
20     UINT kiss_of_death_handler(NX_SNTP_CLIENT *client_ptr, UINT KOD_code);
21
22
23     /* Set up client thread and network resources. */
24
25     NX_PACKET_POOL    client_packet_pool;
26     NX_IP              client_ip;
27     TX_THREAD         demo_client_thread;
28     NX_SNTP_CLIENT    demo_client;
29
30
31
32     /* Configure the SNTP Client to use unicast SNTP. */
33     #define USE_UNICAST
34
35
36     #define CLIENT_IP_ADDRESS    IP_ADDRESS(192,2,2,66)
37     #define SERVER_IP_ADDRESS    IP_ADDRESS(192,2,2,92)
38     #define SERVER_IP_ADDRESS_2  SERVER_IP_ADDRESS
39
40     /* Set up the SNTP network and address index; */
41     UINT    iface_index = 0;
42     UINT    prefix = 64;
43     UINT    address_index;
44
45     /* Set up client thread entry point. */
46     void    demo_client_thread_entry(ULONG info);
47
48     /* Define main entry point. */
49     int main()
50     {
51         /* Enter the ThreadX kernel. */
52         tx_kernel_enter();
53         return 0;
54     }
55
56     /* Define what the initial system looks like. */
57     void tx_application_define(void *first_unused_memory)
58     {
59

```

```

60  UINT    status;
61  UCHAR   *free_memory_pointer;
62
63
64      free_memory_pointer = (UCHAR *)first_unused_memory;
65
66      /* Create client packet pool. */
67      status = nx_packet_pool_create(&client_packet_pool, "SNTP Client Packet Pool",
68                                     NX_SNTP_CLIENT_PACKET_SIZE, free_memory_pointer,
69                                     NX_SNTP_CLIENT_PACKET_POOL_SIZE);
70
71      /* Check for errors. */
72      if (status != NX_SUCCESS)
73      {
74
75          return;
76      }
77
78      /* Initialize the NetX system. */
79      nx_system_initialize();
80
81      /* Update pointer to unallocated (free) memory. */
82      free_memory_pointer = free_memory_pointer + NX_SNTP_CLIENT_PACKET_POOL_SIZE;
83
84      /* Create Client IP instances */
85      status = nx_ip_create(&client_ip, "SNTP IP Instance", CLIENT_IP_ADDRESS,
86                           0xFFFFFFFFUL, &client_packet_pool, _nx_ram_network_driver,
87                           free_memory_pointer, 2048, 1);
88
89      /* Check for error. */
90      if (status != NX_SUCCESS)
91      {
92
93          return;
94      }
95
96      free_memory_pointer = free_memory_pointer + 2048;
97
98      /* Enable ARP and supply ARP cache memory. */
99      status = nx_arp_enable(&client_ip, (void **) free_memory_pointer, 2048);
100
101      /* Check for error. */
102      if (status != NX_SUCCESS)
103      {
104
105          return;
106      }
107
108      /* Update pointer to unallocated (free) memory. */
109      free_memory_pointer = free_memory_pointer + 2048;
110
111      /* Enable UDP for client. */
112      status = nx_udp_enable(&client_ip);
113
114      /* Check for error. */
115      if (status != NX_SUCCESS)

```

```

116     {
117
118         return;
119     }
120
121     status = nx_icmp_enable(&client_ip);
122
123     /* Check for error. */
124     if (status != NX_SUCCESS)
125     {
126
127         return;
128     }
129     /* Create the client thread */
130     status = tx_thread_create(&demo_client_thread, "SNTP Client Thread", demo_client_thread_entry,
131                             (ULONG)(&demo_client), free_memory_pointer, 2048,
132                             4, 4, TX_NO_TIME_SLICE, TX_DONT_START);
133
134     /* Check for errors */
135     if (status != TX_SUCCESS)
136     {
137
138         return;
139     }
140
141     /* Update pointer to unallocated (free) memory. */
142     free_memory_pointer = free_memory_pointer + 2048;
143
144     /* set the SNTP network interface to the primary interface. */
145     iface_index = 0;
146
147     /* Create the SNTP Client to run in broadcast mode.. */
148     status = nx_snmp_client_create(&demo_client, &client_ip, iface_index, &client_packet_pool,
149                                  leap_second_handler,
150                                  kiss_of_death_handler,
151                                  NULL /* no random_number_generator callback */);
152
153     /* Check for error. */
154     if (status != NX_SUCCESS)
155     {
156
157         /* Bail out! */
158         return;
159     }
160
161     tx_thread_resume(&demo_client_thread);
162
163     return;
164 }
165
166 /* Define size of buffer to display client's local time. */
167 #define BUFSIZE 50
168
169 /* Define the client thread. */
170 void demo_client_thread_entry(ULONG info)
171 {

```



```

172
173  UINT  status;
174  UINT  spin;
175  UINT  server_status;
176  CHAR  buffer[BUFSIZE];
177  ULONG base_seconds;
178  ULONG base_fraction;
179  ULONG seconds, milliseconds;
180
181
182  /* Give other threads (IP instance) a chance to initialize. */
183  tx_thread_sleep(100);
184
185
186
187  /* Set up client time updates depending on mode. */
188  #ifdef USE_UNICAST
189
190  /* Initialize the Client for unicast mode to poll the SNTP server once an hour. */
191  /* Use the IPv4 service to set up the Client and set the IPv4 SNTP server. */
192  status = nx_sntp_client_initialize_unicast(&demo_client, SERVER_IP_ADDRESS);
193
194
195  #else /* Broadcast mode */
196
197  /* Initialize the Client for broadcast mode, no roundtrip calculation required and a broadcast SNTP service.
198  */
199  /* Use the IPv4 service to initialize the Client and set IPv4 SNTP broadcast address. */
200  status = nx_sntp_client_initialize_broadcast(&demo_client, NX_NULL, SERVER_IP_ADDRESS);
201  #endif /* USE_UNICAST */
202
203
204  /* Check for error. */
205  if (status != NX_SUCCESS)
206  {
207
208      return;
209  }
210
211  /* Set the base time which is approximately the number of seconds since the turn of the last century.
212  If this is not available in SNTP format, the nx_sntp_client_utility_add_msecs_to_ntp_time service
213  can convert milliseconds to fraction. For how to compute NTP seconds from real time, read the
214  NetX SNTP User Guide.
215
216  Otherwise set the base time to zero and set NX_SNTP_CLIENT_IGNORE_MAX_ADJUST_STARTUP to
217  NX_TRUE for
218  the SNTP Client to accept the first time update without applying a minimum or maximum adjustment
219  parameters (NX_SNTP_CLIENT_MIN_TIME_ADJUSTMENT and
220  NX_SNTP_CLIENT_MAX_TIME_ADJUSTMENT). */
221  base_seconds = 0xd2c96b90; /* Jan 24, 2012 UTC */
222  base_fraction = 0xa132db1e;
223
224  /* Apply to the SNTP Client local time. */
225  status = nx_sntp_client_set_local_time(&demo_client, base_seconds, base_fraction);

```

```

225
226     /* Check for error. */
227     if (status != NX_SUCCESS)
228     {
229
230         return;
231     }
232
233     /* Run whichever service the client is configured for. */
234     #ifdef USE_UNICAST
235
236         status = nx_sntp_client_run_unicast(&demo_client);
237     #else
238         status = nx_sntp_client_run_broadcast(&demo_client);
239     #endif /* USE_UNICAST */
240
241     if (status != NX_SUCCESS)
242     {
243         return;
244     }
245
246     spin = NX_TRUE;
247
248     /* Now check periodically for time changes. */
249     while(spin)
250     {
251
252         /* First verify we have a valid SNTP service running. */
253         status = nx_sntp_client_receiving_updates(&demo_client, &server_status);
254
255         if ((status == NX_SUCCESS) && (server_status == NX_TRUE))
256         {
257
258             /* Server status is good. Now get the Client local time. */
259
260             /* Display the local time in years, months, date format. */
261             status = nx_sntp_client_get_local_time(&demo_client, &seconds, &milliseconds, &buffer[0]);
262
263             if (status == NX_SUCCESS)
264             {
265                 printf("Date: %s\n", &buffer[0]);
266             }
267
268             /* Wait a while before the next update. */
269             tx_thread_sleep(300);
270
271             memset(&buffer[0], 0, BUFSIZE);
272         }
273         else
274         {
275
276             /* Wait a short bit to check again. */
277             tx_thread_sleep(100);
278         }
279     }

```

```

280
281     /* We can stop the SNTP service if for example we think the SNTP service has stopped. */
282     status = nx_sntp_client_stop(&demo_client);
283
284     if (status != NX_SUCCESS)
285     {
286         return;;
287     }
288
289     /* Set up another server and reinitialize the SNTP Client. */
290 #ifdef USE_UNICAST
291     /* Initialize the Client for unicast mode to poll the SNTP server once an hour. */
292     status = nx_sntp_client_initialize_unicast(&demo_client, SERVER_IP_ADDRESS_2);
293
294     /* Check for error. */
295     if (status != NX_SUCCESS)
296     {
297         return;
298     }
299
300     /* Now start the SNTP Client task back up. */
301     status = nx_sntp_client_run_unicast(&demo_client);
302
303     if (status != NX_SUCCESS)
304     {
305         return;
306     }
307 }
308 #else /* Start Client in broadcast */
309
310     status = nx_sntp_client_initialize_broadcast(&demo_client, NX_NULL, SERVER_IP_ADDRESS_2);
311
312     if (status != NX_SUCCESS)
313     {
314         return;
315     }
316
317     /* Now start the SNTP Client task back up. */
318     status = nx_sntp_client_run_broadcast(&demo_client);
319
320     /* Check for error. */
321     if (status != NX_SUCCESS)
322     {
323         return;
324     }
325 #endif
326
327     spin = NX_TRUE;
328
329     /* Now check periodically for time changes. */
330     while(spin)
331     {
332
333         /* First verify we have a valid SNTP service running. */
334         status = nx_sntp_client_receiving_updates(&demo_client, &server_status);
335

```

```

336     if ((status == NX_SUCCESS) && (server_status == NX_TRUE))
337     {
338
339         /* Server status is good. Now retrieve the Client local time. */
340
341         /* Display the local time in years, months, date format. */
342         status = nx_sntp_client_get_local_time(&demo_client, &seconds, &milliseconds, &buffer[0]);
343         if (status == NX_SUCCESS)
344         {
345             printf("Date: %s\n", &buffer[0]);
346         }
347
348         /* It will be a bit longer till the next update. */
349         tx_thread_sleep(200);
350
351         memset(&buffer[0], 0, BUFSIZE);
352     }
353
354     /* Wait a short bit and try again. */
355     tx_thread_sleep(100);
356 }
357
358 /* To return resources to NetX and ThreadX stop the SNTP client and delete the client instance. */
359 status = nx_sntp_client_delete(&demo_client);
360
361 return;
362 }
363
364
365 /* This application defined handler for handling an impending leap second is not
366    required by the SNTP Client. The default handler below only logs the event for
367    every time stamp received with the leap indicator set. */
368
369 UINT leap_second_handler(NX_SNTP_CLIENT *client_ptr, UINT leap_indicator)
370 {
371
372     /* Handle the leap second handler... */
373
374     return NX_SUCCESS;
375 }
376
377 /* This application defined handler for handling a Kiss of Death packet is not
378    required by the SNTP Client. A KOD handler should determine
379    if the Client task should continue vs. abort sending/receiving time data
380    from its current time server, and if aborting if it should remove
381    the server from its active server list.
382
383    Note that the KOD list of codes is subject to change. The list
384    below is current at the time of this software release. */
385
386 UINT kiss_of_death_handler(NX_SNTP_CLIENT *client_ptr, UINT KOD_code)
387 {
388
389     UINT    remove_server_from_list = NX_FALSE;
390     UINT    status = NX_SUCCESS;
391

```

```

392
393 /* Handle kiss of death by code group. */
394 switch (KOD_code)
395 {
396
397     case NX_SNTP_KOD_RATE:
398     case NX_SNTP_KOD_NOT_INIT:
399     case NX_SNTP_KOD_STEP:
400
401         /* Find another server while this one is temporarily out of service. */
402         status = NX_SNTP_KOD_SERVER_NOT_AVAILABLE;
403
404         break;
405
406     case NX_SNTP_KOD_AUTH_FAIL:
407     case NX_SNTP_KOD_NO_KEY:
408     case NX_SNTP_KOD_CRYPT_FAIL:
409
410         /* These indicate the server will not service client with time updates
411            without successful authentication. */
412
413
414         remove_server_from_list = NX_TRUE;
415
416         break;
417
418     default:
419
420
421         /* All other codes. Remove server before resuming time updates. */
422
423         remove_server_from_list = NX_TRUE;
424         break;
425 }
426
427 /* Removing the server from the active server list? */
428 if (remove_server_from_list)
429 {
430
431     /* Let the caller know it has to bail on this server before resuming service. */
432     status = NX_SNTP_KOD_REMOVE_SERVER;
433 }
434
435 return status;
436 }

```

Figure 1 Example of using SNTP Client with NetX

## Configuration Options

There are several configuration options for defining the NetX SNTP Client. The following list describes each in detail:

### Define

### Meaning

**NX\_SNTP\_CLIENT\_THREAD\_STACK\_SIZE**

This option sets the size of the Client thread stack. The default NetX SNTP Client size is 2048.

**NX\_SNTP\_CLIENT\_THREAD\_TIME\_SLICE**

This option sets the time slice of the scheduler allows for Client thread execution. The default NetX SNTP Client size is TX\_NO\_TIME\_SLICE.

**NX\_SNTP\_CLIENT\_THREAD\_PRIORITY** This option sets the Client thread priority. The NetX SNTP Client default value is 2.

**NX\_SNTP\_CLIENT\_PREEMPTION\_THRESHOLD**

This option sets the sets the level of priority at which the Client thread allows preemption. The default NetX SNTP Client value is set to NX\_SNTP\_CLIENT\_THREAD\_PRIORITY.

**NX\_SNTP\_CLIENT\_UDP\_SOCKET\_NAME**

This option sets the UDP socket name. The NetX SNTP Client UDP socket name default is "SNTP Client socket."

**NX\_SNTP\_CLIENT\_UDP\_PORT**

This sets the port which the Client socket is bound to. The default NetX SNTP Client port is 123.

**NX\_SNTP\_SERVER\_UDP\_PORT**

This is port which the Client sends SNTP messages to the SNTP Server on. The default NetX SNTP Server port is 123.

**NX\_SNTP\_CLIENT\_TIME\_TO\_LIVE**

Specifies the number of routers a Client packet can pass before it is discarded. The default NetX SNTP Client is set to 0x80.

**NX\_SNTP\_CLIENT\_MAX\_QUEUE\_DEPTH**

Maximum number of UDP packets (datagrams) that can be queued in the NetX SNTP Client socket. Additional packets received mean the oldest packets are released. The default NetX SNTP Client is set to 5.

**NX\_SNTP\_CLIENT\_PACKET\_TIMEOUT**

Time out for NetX packet allocation. The default NetX SNTP Client packet timeout is 1 second.

**NX\_SNTP\_CLIENT\_NTP\_VERSION**

SNTP version used by the Client  
The NetX SNTP Client API was based on Version 4. The default value is 3.

**NX\_SNTP\_CLIENT\_MIN\_NTP\_VERSION**

Oldest SNTP version the Client will be able to work with. The NetX SNTP Client default is Version 3.

**NX\_SNTP\_CLIENT\_MIN\_SERVER\_STRATUM**

The lowest level (highest numeric stratum level) SNTP Server stratum the Client will accept. The NetX SNTP Client default is 2.

**NX\_SNTP\_CLIENT\_MIN\_TIME\_ADJUSTMENT**

The minimum time adjustment in milliseconds the Client will make to its local clock time. Time adjustments below this will be ignored. The NetX SNTP Client default is 10.

**NX\_SNTP\_CLIENT\_MAX\_TIME\_ADJUSTMENT**

The maximum time adjustment in milliseconds the Client will make to its local clock time. For time adjustments above this amount, the local clock adjustment is limited to the maximum time adjustment. The NetX SNTP Client default is 180000 (3 minutes).

**NX\_SNTP\_CLIENT\_IGNORE\_MAX\_ADJUST\_STARTUP**

This enables the maximum time adjustment to be waived when the Client receives the first update from its time server. Thereafter, the maximum time adjustment is enforced. The intention is to get the Client in synch with the server clock as soon as possible. The NetX SNTP Client default is NX\_TRUE.

**NX\_SNTP\_CLIENT\_MAX\_TIME\_LAPSE**

Maximum allowable amount of time (seconds) elapsed without a valid time update received by the SNTP Client. The SNTP Client will continue in operation but the SNTP Server status is set to NX\_FALSE. The default value is 7200.

.

**NX\_SNTP\_UPDATE\_TIMEOUT\_INTERVAL**

The interval (seconds) at which the SNTP Client timer updates the SNTP Client time remaining since the last valid update received, and the unicast Client updates the poll interval time remaining before sending the next SNTP update request. The default value is 1.

**NX\_SNTP\_CLIENT\_UNICAST\_POLL\_INTERVAL**

The starting poll interval (seconds) on which the Client sends a unicast request to its SNTP server. The NetX SNTP Client default is 3600.

**NX\_SNTP\_CLIENT\_EXP\_BACKOFF\_RATE**

The factor by which the current Client unicast poll interval is increased. When the Client fails to receive a server time update, or receiving indications from the server that it is temporarily unavailable (e.g. not synchronized yet) for time update



service, it will increase the current poll interval by this rate up to but not exceeding

`NX_SNTP_CLIENT_MAX_TIME_LAPSE`. The default is 2.

#### **`NX_SNTP_CLIENT_RTT_REQUIRED`**

This option if enabled requires that the SNTP Client calculate round trip time of SNTP messages when applying Server updates to the local clock. The default value is `NX_FALSE` (disabled).

#### **`NX_SNTP_CLIENT_MAX_ROOT_DISPERSION`**

The maximum server clock dispersion (microseconds), which is a measure of server clock precision, the Client will accept. To disable this requirement, set the maximum root dispersion to 0x0. The NetX SNTP Client default is set to 50000.

#### **`NX_SNTP_CLIENT_INVALID_UPDATE_LIMIT`**

The limit on the number of consecutive invalid updates received from the Client server in either broadcast or unicast mode. When this limit is reached, the Client sets the current SNTP Server status to invalid (`NX_FALSE`) although it will continue to try to receive updates from the Server. The NetX SNTP Client default is 3.

#### **`NX_SNTP_CLIENT_RANDOMIZE_ON_STARTUP`**

This determines if the SNTP Client in unicast mode should send its first SNTP request with the current SNTP server after a random wait interval. It is used in cases where significant numbers of SNTP Clients are starting up simultaneously to limit traffic congestion on the SNTP Server. The default value is `NX_FALSE`.

<b>NX_SNTP_CLIENT_SLEEP_INTERVAL</b>	The time interval during which the SNTP Client task sleeps. This allows the application API calls to be executed by the SNTP Client. The default value is 1 timer tick.
<b>NX_SNTP_CURRENT_YEAR</b>	Set this value to the current year (same year as in NTP time being evaluated). To display date in year/month/date format, the input date need not be in the same year. The default value is 2015.
<b>NTP_SECONDS_AT_01011999</b>	This is the number of seconds into the first NTP Epoch on the master NTP clock. It is defined as 0xBA368E80. To disable display of NTP seconds into date and time, set to zero.

## Chapter 3

### Description of NetX SNTP Client Services

This chapter contains a description of all NetX SNTP Client 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_sntp_client_create`  
*Create the SNTP Client*

`nx_sntp_client_delete`  
*Delete the SNTP Client*

`nx_sntp_client_get_local_time`  
*Get SNTP Client local time*

`nx_sntp_client_initialize_broadcast`  
*Initialize Client for broadcast operation*

`nx_sntp_client_initialize_unicast`  
*Initialize Client for unicast operation*

`nx_sntp_client_receiving_updates`  
*Client is currently receiving valid SNTP updates*

`nx_sntp_client_run_broadcast`  
*Receive time updates from server*

`nx_sntp_client_run_unicast`  
*Send requests and receive time updates from server*

`nx_sntp_client_set_local_time`  
*Set SNTP Client initial local time*

`nx_sntp_client_stop`  
*Stop the SNTP Client thread*

`nx_sntp_client_utility_display_date_and_time`

*Display NTP time in seconds*

`nx_sntp_client_utility_msecs_to_fraction`

*Convert milliseconds to NTP fraction component*

## nx\_sntp\_client\_create

---

Create an SNTP Client

### Prototype

```
UINT nx_sntp_client_create(NX_SNTP_CLIENT *client_ptr, NX_IP *ip_ptr,
    UINT iface_index, NX_PACKET_POOL *packet_pool_ptr,
    UINT (*leap_second_handler)(NX_SNTP_CLIENT *client_ptr, UINT
        indicator),
    UINT (*kiss_of_death_handler)(NX_SNTP_CLIENT *client_ptr,
        NX_SNTP_TIME_MESSAGE *server_time_msg),
    VOID (random_number_generator)(struct NX_SNTP_CLIENT_STRUCT
        *client_ptr, ULONG *rand));
```

### Description

This service creates an SNTP Client instance.

### Input Parameters

<b>client_ptr</b>	Pointer to SNTP Client control block
<b>ip_ptr</b>	Pointer to Client IP instance
<b>iface_index</b>	Index to SNTP network interface
<b>packet_pool_ptr</b>	Pointer to Client packet pool
<b>leap_second_handler</b>	Callback for application response to impending leap second
<b>kiss_of_death_handler</b>	Callback for application response to receiving Kiss of Death packet
<b>random_number_generator</b>	Callback to random number generator service

### Return Values

<b>NX_SUCCESS</b>	(0x00) Successful Client creation
<b>NX_SNTP_INSUFFICIENT_PACKET_PAYLOAD</b>	(0xD2A) Invalid non pointer input
<b>NX_PTR_ERROR</b>	(0x07) Invalid pointer input
<b>NX_INVALID_INTERFACE</b>	(0x4C) Invalid network interface

## Allowed From

Initialization, Threads

## Example

```
/* Create the SNTP Client on the primary interface. */
UINT iface_index = 0;
status = nx_sntp_client_create(&demo_client, iface_index, &client_ip,
                               &client_packet_pool, leap_second_handler,
                               kiss_of_death_handler,
                               NULL /* no random_number_generator callback */);

/* If status is NX_SUCCESS an SNTP Client instance was successfully
   created. */
```

## nx\_sntp\_client\_delete

---

Delete an SNTP Client

### Prototype

```
UINT nx_sntp_client_delete(NX_SNTP_CLIENT *client_ptr);
```

### Description

This service deletes an SNTP Client instance.

### Input Parameters

<b>client_ptr</b>	Pointer to SNTP Client control block
-------------------	--------------------------------------

### Return Values

<b>NX_SUCCESS</b>	(0x00) Successful Client creation
<b>NX_CALLER_ERROR</b>	(0x11) Invalid caller of service
<b>NX_PTR_ERROR</b>	(0x07) Invalid pointer input

### Allowed From

Threads

### Example

```
/* Delete the SNTP Client. */
status = nx_sntp_client_delete(&demo_client);

/* If status is NX_SUCCESS an SNTP Client instance was successfully
   deleted. */
```

## nx\_sntp\_client\_get\_local\_time

---

Get the SNTP Client local time

### Prototype

```
UINT nx_sntp_client_get_local_time(NX_SNTP_CLIENT *client_ptr ,
                                   ULONG *seconds,
                                   ULONG *milliseconds,
                                   CHAR *buffer);
```

### Description

This service gets the SNTP Client local time with an option buffer pointer input to receive the data in string message format.

### Input Parameters

<b>client_ptr</b>	Pointer to SNTP Client control block
<b>seconds</b>	Pointer to local time seconds
<b>milliseconds</b>	Pointer to milliseconds component
<b>buffer</b>	Pointer to buffer to write time data

### Return Values

<b>NX_SUCCESS</b>	(0x00) Successful Client creation
<b>NX_CALLER_ERROR</b>	(0x11) Invalid caller of service
<b>NX_PTR_ERROR</b>	(0x07) Invalid pointer input

### Allowed From

Threads

### Example

```
/* Get the SNTP Client local time without the string message option. */
ULONG base_seconds;
ULONG base_milliseconds;

status = nx_sntp_client_get_local_time(&demo_client, &base_seconds,
                                       &base_milliseconds, NX_NULL);

/* If status is NX_SUCCESS an SNTP Client time was successfully
   retrieved. */
```



## Prototype

### Description

## Input Parameters

<b>client_ptr</b>	Pointer to SNTP Client control block
<b>multicast_server_address</b>	SNTP multicast address
<b>broadcast_time_server</b>	SNTP server broadcast address

<b>NX_SUCCESS</b>	(0x00)	Successful Client Creation
<b>NX_INVALID_PARAMETERS</b>	(0x4D)	Invalid non pointer input
<b>NX_PTR_ERROR</b>	(0x07)	Invalid pointer input
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of service

## Initialization, Threads

[illegible]

```
/* If status is NX_SUCCESS the Client was successfully initialized. */
```

## **nx\_snmp\_client\_initialize\_unicast**

---

Set up the SNMP Client to run in unicast

### **Prototype**

```
UINT nx_snmp_client_initialize_unicast(NX_SNMP_CLIENT * client_ptr,
                                       ULONG unicast_time_server);
```

### **Description**

This service initializes the Client for unicast operation by setting the SNMP Server IP address and initializing SNMP startup parameters and timeouts.

### **Input Parameters**

<b>client_ptr</b>	Pointer to SNMP Client control block
<b>unicast_time_server</b>	SNMP server IP address

### **Return Values**

<b>NX_SUCCESS</b>	(0x00) Client successfully initialized
<b>NX_INVALID_PARAMETERS</b>	(0x4D) Invalid non pointer input
<b>NX_PTR_ERROR</b>	(0x07) Invalid pointer input
<b>NX_CALLER_ERROR</b>	(0x11) Invalid caller of service

### **Allowed From**

Initialization, Threads

### **Example**

```
/* Initialize the Client for unicast operation. */
status = nx_snmp_client_initialize_unicast(&client_ptr, IP_ADDRESS(192,2,2,1));

/* If status is NX_SUCCESS the Client is initialized for unicast operation. */
```

## **nx\_sntp\_client\_receiving\_updates**

---

Indicate if Client is receiving valid updates

### **Prototype**

```
UINT nx_sntp_client_receiving_updates(NX_SNTP_CLIENT *client_ptr,
                                       UINT *receive_status);
```

### **Description**

This service indicates if the Client is receiving valid SNTP updates. If the maximum time lapse without a valid update or limit on consecutive invalid updates is exceeded, the receive status is returned as false. Note that the SNTP Client is still running and if the application wishes to restart the SNTP Client with another unicast or broadcast/multicast server it must stop the SNTP Client using the *nx\_sntp\_client\_stop* service, reinitialize the Client using one of the initialize services with another server.

### **Input Parameters**

<b>client_ptr</b>	Pointer to SNTP Client control block.
<b>receive_status</b>	Pointer to indicator if Client is receiving valid updates.

### **Return Values**

<b>NX_SUCCESS</b>	(0x00) Client successfully received update status
<b>NX_PTR_ERROR</b>	(0x07) Invalid pointer input

### **Allowed From**

Initialization, Threads

### **Example**

```
/* Determine if the SNTP Client is receiving valid updates. */
UINT receive_status;

status = nx_sntp_client_receiving_updates(client_ptr, &receive_status);

/* If status is NX_SUCCESS and receive_status is NX_TRUE, the client is
   currently receiving valid updates. */
```

## nx\_sntp\_client\_request\_unicast\_time

---

Send a unicast request directly to the NTP Server

### Prototype

```
UINT nx_sntp_client_request_unicast_time(NX_SNTP_CLIENT *client_ptr,
                                         UINT wait_option);
```

### Description

This service allows the application to directly send a unicast request to the NTP server asynchronously from the SNTP Client thread task. The wait option specifies how long to wait for a response. If successful, the application can use other SNTP Client services to obtain the latest time. See section **SNTP Asynchronous Unicast Requests** for more details.

### Input Parameters

<b>client_ptr</b>	Pointer to SNTP Client control block.
<b>Wait_option</b>	Wait option for NTP response in timer ticks.

### Return Values

<b>NX_SUCCESS</b>	(0x00)	Client successfully sends and receives unicast update
<b>NX_SNTP_CLIENT_NOT_STARTED</b>	(0xD0B)	Client thread not started
<b>NX_PTR_ERROR</b>	(0x07)	Invalid pointer input
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of service

### Allowed From

Threads

### Example

```
/* Determine if the SNTP Client is receiving valid updates. */
UINT receive_status;

status = nx_sntp_client_request_unicast_time(client_ptr, 400);

/* If status is NX_SUCCESS and receive_status is NX_TRUE, the client is received
   a valid response to the unicast request. */
```

## **nx\_sntp\_client\_run\_broadcast**

---

Run the Client in broadcast mode

### **Prototype**

```
UINT nx_sntp_client_run_broadcast(NX_SNTP_CLIENT *client_ptr);
```

### **Description**

This service starts the Client in broadcast mode where it will wait to receive broadcasts from the SNTP server. If a valid broadcast SNTP message is received, the SNTP client timeout for maximum lapse without an update and count of consecutive invalid messages received are reset. If either of these limits are exceeded, the SNTP Client sets the server status to invalid although it will still wait to receive updates. The application can poll the SNTP Client task for server status, and if invalid stop the SNTP Client and reinitialize it with another SNTP broadcast address. It can also switch to a unicast SNTP server.

### **Input Parameters**

<b>client_ptr</b>	Pointer to SNTP Client control block.
-------------------	---------------------------------------

### **Return Values**

<b>NX_SNTP_CLIENT_ALREADY_STARTED</b>	(0xD0C) Client already started
<b>NX_SNTP_CLIENT_NOT_INITIALIZED</b>	(0xD01) Client not initialized
<b>NX_PTR_ERROR</b>	(0x07) Invalid pointer input
<b>NX_CALLER_ERROR</b>	(0x11) Invalid caller of service

### **Allowed From**

Threads

### **Example**

```
/* Start Client running in broadcast mode. */
status = nx_sntp_client_run_broadcast(client_ptr);

/* If status is NX_SUCCESS, the client is successfully started. */
```

## **nx\_sntp\_client\_run\_unicast**

---

Run the Client in unicast mode

### **Prototype**

```
UINT nx_sntp_client_run_unicast(NX_SNTP_CLIENT *client_ptr);
```

### **Description**

This service starts the Client in unicast mode where it periodically sends a unicast request to its SNTP Server for a time update. If a valid SNTP message is received, the SNTP client timeout for maximum lapse without an update, initial polling interval and count of consecutive invalid messages received are reset. If the either of these limits are exceeded, the SNTP Client sets the Server status to invalid although it will still poll and wait to receive updates. The application can poll the SNTP Client task for server status, and if invalid stop the SNTP Client and reinitialize it with another SNTP unicast address. It can also switch to a broadcast SNTP server.

.

### **Input Parameters**

<b>client_ptr</b>	Pointer to SNTP Client control block.
-------------------	---------------------------------------

### **Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successfully started Client in unicast mode
<b>Status</b>	-----	Internal error occurred
<b>NX_SNTP_CLIENT_ALREADY_STARTED</b>	(0xD0C)	Client already started
<b>NX_SNTP_CLIENT_NOT_INITIALIZED</b>	(0xD01)	Client not initialized
<b>NX_PTR_ERROR</b>	(0x07)	Invalid pointer input
<b>NX_CALLER_ERROR</b>	(0x11)	Invalid caller of service





**Allowed From**

Threads

**Example**

```
/* Start the Client in unicast mode. */  
status = nx_snmp_client_run_unicast(client_ptr);  
/* If status = NX_SUCCESS, the Client was successfully started. */
```

## nx\_sntp\_client\_set\_local\_time

---

Set the SNTP Client local time

### Prototype

```
UINT nx_sntp_client_set_local_time(NX_SNTP_CLIENT *client_ptr,
                                   ULONG seconds, ULONG fraction);
```

### Description

This service sets the SNTP Client local time with the input time, in SNTP format e.g. seconds and 'fraction' which is the format for putting fractions of a second in hexadecimal format. It is intended for use when starting up the SNTP Client to give it a base time upon which to compare received updates for valid time data. This is optional; the SNTP Client can run without a starting local time. Input time candidates can be obtained from existing SNTP time values (on the Internet) and are computed as the number of seconds since January 1, 1900 (until 2036 when a new 'epoch' will be started).

### Input Parameters

<b>client_ptr</b>	Pointer to SNTP Client control block
<b>seconds</b>	Seconds component of the time input
<b>fraction</b>	Subseconds component in the SNTP fraction format

### Return Values

<b>NX_SUCCESS</b>	(0x00) Successfully set local time
<b>NX_PTR_ERROR</b>	(0x07) Invalid pointer input

### Allowed From

Initialization

### Example

```
/* Set the SNTP Client local time. */
base_seconds = 0xd2c50b71;
base_fraction = 0xa132db1e;

status = nx_sntp_client_set_local_time(&demo_client, base_seconds,
                                       base_fraction);

/* If status is NX_SUCCESS an SNTP Client time was successfully
   set. */
```

## nx\_sntp\_client\_set\_time\_update\_notify

---

Set the SNTP update callback

### Prototype

```
UINT nx_sntp_client_set_time_update_notify(NX_SNTP_CLIENT *client_ptr,
      VOID (time_update_cb)(NX_SNTP_TIME_MESSAGE
                           *time_update_ptr, NX_SNTP_TIME *local_time)));
```

### Description

This service sets callback to notify the application when the SNTP Client receives a valid time update. It supplies the actual SNTP message and the SNTP Client's local time (usually the same) in NTP format. The application can use the NTP data directly or call the *nx\_sntp\_client\_utility\_display\_date\_time service* to convert the time to human readable format.

### Input Parameters

<b>client_ptr</b>	Pointer to SNTP Client control block
<b>time_update_cb</b>	Pointer to callback function

### Return Values

<b>NX_SUCCESS</b>	(0x00) Successfully set callback
<b>NX_PTR_ERROR</b>	(0x07) Invalid pointer input

### Allowed From

Initialization

### Example

```
/* Set the SNTP Client time update callback. */
VOID client_time_update_notify(NX_SNTP_TIME_MESSAGE *time_update_ptr,
                              NX_SNTP_TIME *local_time);

NX_SNTP_CLIENT demo_client;

status = nx_sntp_client_set_time_update_notify(&demo_client, time_update_cb);

/* If status is NX_SUCCESS an SNTP Client time update callback was successfully
   set. */
```

## nx\_sntp\_client\_stop

---

Stop the SNTP Client thread

### Prototype

```
UINT nx_sntp_client_stop(NX_SNTP_CLIENT *client_ptr);
```

### Description

This service stops the SNTP Client thread. The SNTP Client thread tasks, which runs in an infinite loop, pauses on every iteration to release control of the SNTP Client state and allow applications to make API calls on the SNTP Client.

### Input Parameters

<b>client_ptr</b>	Pointer to SNTP Client control block
-------------------	--------------------------------------

### Return Values

<b>NX_SUCCESS</b>	(0x00) Successful stopped Client thread
<b>NX_SNTP_CLIENT_NOT_STARTED</b>	(0xD) SNTP Client thread not started
<b>NX_PTR_ERROR</b>	(0x07) Invalid pointer input

### Allowed From

Initialization, Threads

### Example

```
/* Stop the SNTP Client. */
status = nx_sntp_client_stop(&demo_client);

/* If status is NX_SUCCESS an SNTP Client instance was successfully
   stopped. */
```

## **nx\_sntp\_client\_utility\_display\_date\_time**

---

Convert an NTP Time to Date and Time string

### **Prototype**

```
UINT nx_sntp_client_utility_display_date_time (NX_SNTP_CLIENT
                                              *client_ptr, CHAR *buffer, UINT length);
```

### **Description**

This service converts the SNTP Client local time to a year month date format and returns the date in the supplied buffer. It requires that the NX\_SNTP\_CURRENT\_YEAR be set. The SNTP Client local time need not be in the same year as NX\_SNTP\_CURRENT\_YEAR, but NX\_SNTP\_CURRENT\_YEAR must be set or the function returns an error.

### **Input Parameters**

<b>client_ptr</b>	Pointer to SNTP Client
<b>buffer</b>	Pointer to buffer to store date
<b>length</b>	Size of input buffer

### **Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successful conversion
<b>status</b>	-----	Internal error occurred
<b>NX_SNTP_ERROR_CONVERTING_DATETIME</b>	(0xD08)	NX_SNTP_CURRENT_YEAR not defined or no local client time established
<b>NX_SNTP_INVALID_DATETIME_BUFFER</b>	(0xD07)	Insufficient buffer length

## Allowed From

Initialization, Threads

## Example

```
/* Convert and display the Client's local time. */  
status = nx_sntp_client_utility_display_date_time(client_ptr, buffer,  
                                                  sizeof(buffer));  
/* If status is NX_SUCCESS, date was successfully written to buffer. */
```

## **nx\_sntp\_client\_utility\_msecs\_to\_fraction**

---

Convert milliseconds to an NTP fraction component

### **Prototype**

```
UINT nx_sntp_client_utility_msecs_to_fraction (ULONG milliseconds,
                                              ULONG *fraction);
```

### **Description**

This service converts the input milliseconds to the NTP fraction component. It is intended for use with applications that have a starting base time for the SNTP Client but not in NTP seconds/fraction format. The number of milliseconds must be less than 1000 to make a valid fraction.

### **Input Parameters**

<b>milliseconds</b>	Milliseconds to convert
<b>fraction</b>	Pointer to milliseconds converted to fraction

### **Return Values**

<b>NX_SUCCESS</b>	(0x00)	Successful conversion
<b>NX_SNTP_OVERFLOW_ERROR</b>	(0xD32)	Error converting time to a date
<b>NX_SNTP_INVALID_TIME</b>	(0xD30)	Invalid SNTP data input

### **Allowed From**

Initialization, Threads

### **Example**

```
/* Convert the milliseconds to a fraction. */

status = nx_sntp_client_utility_msecs_to_fraction(milliseconds, &fraction);

/* If status is NX_SUCCESS, data was successfully converted. */
```

## Appendix A: SNTP Fatal Error Codes

The following error codes will result in the SNTP Client aborting time updates with the current server. It is up to the application to decide if the server should be removed from the SNTP Client list of available servers, or simply switch to the next available server on the list. The definition of each error status is defined in *nx\_sntp\_client.h*.

When the SNTP Client returns an error from the list below to the application, the Server should probably be replaced with another Server. Note that the `NX_SNTP_KOD_REMOVE_SERVER` error status is left to the SNTP Client kiss of death handler (callback function) to set:

<code>NX_SNTP_KOD_REMOVE_SERVER</code>	<code>0xD0C</code>
<code>NX_SNTP_SERVER_AUTH_FAIL</code>	<code>0xD0D</code>
<code>NX_SNTP_INVALID_NTP_VERSION</code>	<code>0xD11</code>
<code>NX_SNTP_INVALID_SERVER_MODE</code>	<code>0xD12</code>
<code>NX_SNTP_INVALID_SERVER_STRATUM</code>	<code>0xD15</code>

When the SNTP Client returns an error from the list below to the application, the Server may only temporarily be unable to provide valid time updates and need not be removed:

<code>NX_SNTP_NO_UNICAST_FROM_SERVER</code>	<code>0xD09</code>
<code>NX_SNTP_SERVER_CLOCK_NOT_SYNC</code>	<code>0xD0A</code>
<code>NX_SNTP_KOD_SERVER_NOT_AVAILABLE</code>	<code>0xD0B</code>
<code>NX_SNTP_OVER_BAD_UPDATE_LIMIT</code>	<code>0xD17</code>
<code>NX_SNTP_BAD_SERVER_ROOT_DISPERSION</code>	<code>0xD16</code>
<code>NX_SNTP_INVALID_RTT_TIME</code>	<code>0xD21</code>
<code>NX_SNTP_KOD_SERVER_NOT_AVAILABLE</code>	<code>0xD24</code>



---

NetX Simple Network Time Protocol (SNTP) Client 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 Simple Network Time Protocol (SNTP) Client User Guide