NetX Duo 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

Part Number: 000-1052
Revision 5.0

Contents

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 ms, 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 Duo SNTP Client Requirements

The NetX Duo 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 whatever UDP port their broadcast server is sending on, usually 123. The NetX Duo SNTP Client host application must have one or more IP SNTP Server addresses.

NetX Duo 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 If that Server appears to be no longer valid, the host application must stop the SNTP Client task, and reinitialize it with another SNTP server address, either broadcast or unicast.

The SNTP Client does not support manycast.

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

NetX Duo 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 Duo 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 includes setting up 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 host 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 to stop the SNTP Client, and one of the initialize services to set another SNTP Server address. To restart the SNTP Client, the host application calls *nx_sntp_client_run_broadcast* or *nx_sntp_client_run_unicast*. Note that the host can change SNTP Client operating mode in the initialize call to switch to unicast or broadcast as desired.

Local Clock Operation

The SNTP time is the number of seconds elapsed since Jan 1 1900. Before the SNTP Client runs, the host 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 host 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 Duo SNTP Client. For a description of NTP time format, see **Appendix B. NTP Time Stamp Format.**

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 host 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 (non NULL). See **Appendix B** for SNTP Time data format. 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 non successful 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.

SNTP and Multi Homed Hosts

Starting with NetX Duo 5.6, NetX Duo supports multi homed hosts.

SNTP and NTP RFCs

NetX Duo SNTP client is compliant with RFC4330 and related RFCs.

Chapter 2

Installation and Use of NetX Duo SNTP Client

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

Product Distribution

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

nxd_sntp_client.c	SNTP Client C source file
nxd_sntp_client.h	SNTP Client Header file
demo_netxduo_sntp_client.c	Demonstration SNTP Client application
nxd_sntp.pdf	NetX Duo SNTP Client User Guide

NetX Duo SNTP Client Installation

In order to use SNTP for NetX Duo, the entire distribution mentioned previously should be copied to the same directory where NetX Duo is installed. For example, if NetX Duo is installed in the directory "*threadx**arm7**green*" then the *nxd_sntp_client.c* and *nxd_sntp_client.h* files should be copied into this directory.

Using NetX Duo SNTP Client

Using NetX Duo SNTP Client is easy. Basically, the application code must include *nxd_sntp_client.h* after it includes *tx_api.h, fx_api.h,* and *nx_api.h*, in order to use ThreadX and NetX Duo, respectively. Once *nxf_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 *nxf_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 Duo SNTP Client.

Note that since the NetX Duo SNTP Client utilizes NetX Duo UDP services, UDP must be enabled with the *nx_udp_enable* call prior to using SNTP services.

Small Example System

An example of how easy it is to use NetX Duo SNTP is described below. In this example, the SNTP include file *nxd_sntp_client.h* is brought in at line 14. The SNTP Client control block "*demo_client* " was defined as a global variable at line 31 previously, and following that SNTP Address information for connecting to an SNTP Server. Next, the SNTP Client is created in "*tx_application_define*" at line 155.

This demo can be used with IPv6 or IPv4. To run the SNTP Client over IPv6, define USE_IPV6 on line 40. IPv6 must be enabled in NetX Duo as well. In lines 196-253, the SNTP Client host is set up for IPv6 address validation and ICMPv6 and IPv6 services from NetX Duo.

Setting a baseline time is optional. The values provided in lines 302 and 303 are taken from a standard NTP server data. These time is applied to the SNTP Client before starting it on line 306 using the *nx_sntp_client_set_local_time* service. This is useful primarily for having a base time to compare the SNTP Client's first received update. Otherwise the SNTP Client accepts the first received update automatically.

From here the SNTP Client can start in broadcast or unicast mode. First it must be initialized (see lines 256-283) for starting SNTP parameters. The 'duo' services e.g. *nxd_sntp_client_initialize_broadcast* and *nxd_sntp_client_initialize_unicast* can take either IPv4 or IPv6 address types, while *the nx_sntp_client_initialize_broadcast* and *nx_sntp_client_initialize_unicast* services will only accept IPv4 address types.

After successful creation, the SNTP Client is started at line 318-320. The host application then spins in loop and periodically checks for updates. It can use the *nx_sntp_client_receiving _updates* to verify that the SNTP Client is currently receiving valid updates. If so, it can retrieve that time using the *nx_sntp_client_get_local_time* service on line 343.

The SNTP Client can be stopped at any time using the *nx_sntp_client_stop* service (line 364) if for example it detects the SNTP Client is no longer receiving valid updates.. To restart the Client, the host application must call either unicast or broadcast initialize and run

services. Note that the SNTP Client can switch SNTP servers and modes (unicast or broadcast) while stopped.

```
/*


   This is a small demo of the NetX Duo SNTP Client on the high-performance NetX
   Duo UDP/IP stack. This demo relies on Thread, NetX Duo and NetX Duo SNTP Client
   API to execute the Simple Network Time Protocol in unicast and broadcast modes.

 */


#include <stdio.h>
#include "nx_api.h"
#include "nx_ip.h"
#include "nxd_sntp_client.h"

/* Set up generic network driver for demo program.  Replace with actual
   ethernet driver to send and receive packets out on the wire. */
VOID   _nx_ram_network_driver(struct NX_IP_DRIVER_STRUCT *driver_req);

/* Optional application defined services of the NetX SNTP Client. */

UINT leap_second_handler(NX_SNTP_CLIENT *client_ptr, UINT leap_indicator);
UINT kiss_of_death_handler(NX_SNTP_CLIENT *client_ptr, UINT KOD_code);


/* Set up client thread and network resources. */

NX_PACKET_POOL      client_packet_pool;
NX_IP               client_ip;
TX_THREAD           demo_client_thread;
NX_SNTP_CLIENT      demo_client;

/* Configure the SNTP Client to use IPv6. If not enabled, the
   Client will use IPv4.  Note: IPv6 must be enabled in NetX Duo
   for the Client to communicate over IPv6.    */
#define USE_IPV6


/* Configure the SNTP Client to use unicast SNTP. */
#define USE_UNICAST


#define CLIENT_IP_ADDRESS       IP_ADDRESS(192,2,2,66)
#define SERVER_IP_ADDRESS       IP_ADDRESS(192,2,2,92)
#define SERVER_IP_ADDRESS_2     SERVER_IP_ADDRESS

/* Set up the SNTP network and address index; */
UINT     iface_index =0;
UINT     prefix = 64;
UINT     address_index;

/* Set up client thread entry point. */
void    demo_client_thread_entry(ULONG info);

/* Define main entry point.  */
int main()
{
    /* Enter the ThreadX kernel.  */
    tx_kernel_enter();
    return 0;
}

/* Define what the initial system looks like.  */
void    tx_application_define(void *first_unused_memory)
{

UINT     status;
```

```
68  UCHAR    *free_memory_pointer;
69
70
71      free_memory_pointer = (UCHAR *)first_unused_memory;
72
73      /* Create client packet pool. */
74      status =  nx_packet_pool_create(&client_packet_pool, "SNTP Client Packet Pool",
75                                      NX_SNTP_CLIENT_PACKET_SIZE, ree_memory_pointer,
76                                      NX_SNTP_CLIENT_PACKET_POOL_SIZE);
77
78      /* Check for errors. */
79      if (status != NX_SUCCESS)
80      {
81
82          return;
83      }
84
85      /* Initialize the NetX system. */
86      nx_system_initialize();
87
88      /* Update pointer to unallocated (free) memory. */
89      free_memory_pointer =  free_memory_pointer + NX_SNTP_CLIENT_PACKET_POOL_SIZE;
90
91      /* Create Client IP instances */
92      status = nx_ip_create(&client_ip, "SNTP IP Instance", CLIENT_IP_ADDRESS,
93                            0xFFFFFF00UL, &client_packet_pool, nx_ram_network_driver,
94                            free_memory_pointer, 2048, 1);
95
96      /* Check for error. */
97      if (status != NX_SUCCESS)
98      {
99
100         return;
101     }
102
103     free_memory_pointer =  free_memory_pointer + 2048;
104
105     /* Enable ARP and supply ARP cache memory. */
106     status =  nx_arp_enable(&client_ip, (void **) free_memory_pointer, 2048);
107
108     /* Check for error. */
109     if (status != NX_SUCCESS)
110     {
111
112         return;
113     }
114
115     /* Update pointer to unallocated (free) memory. */
116     free_memory_pointer = free_memory_pointer + 2048;
117
118     /* Enable UDP for client. */
119     status =  nx_udp_enable(&client_ip);
120
121     /* Check for error. */
122     if (status != NX_SUCCESS)
123     {
124
125         return;
126     }
127
128     status = nx_icmp_enable(&client_ip);
129
130     /* Check for error. */
131     if (status != NX_SUCCESS)
132     {
133
134         return;
135     }
136     /* Create the client thread */
137     status = tx_thread_create(&demo_client_thread, "SNTP Client Thread",
                                  demo_client_thread_entry,
```

```
138                                 (ULONG)(&demo_client), free_memory_pointer, 2048,
139                                  4, 4, TX_NO_TIME_SLICE, TX_DONT_START);
140
141      /* Check for errors */
142      if (status != TX_SUCCESS)
143      {
144
145          return;
146      }
147
148      /* Update pointer to unallocated (free) memory. */
149      free_memory_pointer = free_memory_pointer + 2048;
150
151      /* set the SNTP network interface to the primary interface. */
152      iface_index = 0;
153
154      /* Create the SNTP Client to run in broadcast mode.. */
155      status =  nx_sntp_client_create(&demo_client, &client_ip, iface_index,
                                        &client_packet_pool,
156                                      leap_second_handler,
157                                      kiss_of_death_handler,
158                                      NULL /* no random_number_generator callback
*/);
159
160      /* Check for error. */
161      if (status != NX_SUCCESS)
162      {
163
164          /* Bail out!*/
165          return;
166      }
167
168      tx_thread_resume(&demo_client_thread);
169
170      return;
171  }
172
173  /* Define size of buffer to display client's local time. */
174  #define BUFSIZE 50
175
176  /* Define the client thread.  */
177  void    demo_client_thread_entry(ULONG info)
178  {
179
180  UINT   status;
181  UINT   spin;
182  UINT   server_status;
183  CHAR   buffer[BUFSIZE];
184  ULONG  base_seconds;
185  ULONG  base_fraction;
186  ULONG  seconds, milliseconds;
187  #ifdef USE_IPV6
188  NXD_ADDRESS sntp_server_address, sntp_server_address2;
189  NXD_ADDRESS client_ip_address;
190  #endif
191
192
193      /* Give other threads (IP instance) a chance to initialize. */
194      tx_thread_sleep(100);
195
196  #ifdef USE_IPV6
197      /* Set up IPv6 services. */
198      status = nxd_ipv6_enable(&client_ip);
199
200      status += nxd_icmp_enable(&client_ip);
201
202      if (status  != NX_SUCCESS)
203          return;
204
205      client_ip_address.nxd_ip_address.v6[0] = 0x20010db8;
206      client_ip_address.nxd_ip_address.v6[1] = 0x0000f101;
```

```
207      client_ip_address.nxd_ip_address.v6[2] = 0x0;
208      client_ip_address.nxd_ip_address.v6[3] = 0x101;
209      client_ip_address.nxd_ip_version = NX_IP_VERSION_V6;
210
211      /* Set the IPv6 server address. */
212      sntp_server_address.nxd_ip_address.v6[0] = 0x20010db8;
213      sntp_server_address.nxd_ip_address.v6[1] = 0x0000f101;
214      sntp_server_address.nxd_ip_address.v6[2] = 0x0;
215      sntp_server_address.nxd_ip_address.v6[3] = 0x00000106;
216      sntp_server_address.nxd_ip_version = NX_IP_VERSION_V6;
217
218      /* Set up our 'alternative' time server.  Actually we'll just copy over the
219         server above and present it as an alternative server when we restart the 220
             SNTP Client below. */
220      COPY_NXD_ADDRESS(&sntp_server_address, &sntp_server_address2);
221
222      /* Establish the link local address for the host. The RAM driver creates
223         a virtual MAC address. */
224  #ifdef MULTIHOME_NETXDUO
225      status = nxd_ipv6_address_set(&client_ip, iface_index, NX_NULL, 10, NULL);
226  #else
227      status = nxd_ipv6_linklocal_address_set(&client_ip, NULL);
228  #endif
229
230      /* Check for link local address set error.  */
231      if (status != NX_SUCCESS)
232      {
233          return;
234      }
235
236       /* Set the host global IP address. We are assuming a 64
237         bit prefix here but this can be any value (< 128). */
238  #ifdef MULTIHOME_NETXDUO
239      status = nxd_ipv6_address_set(&client_ip, iface_index, &client_ip_address,
prefix, &address_index);
240  #else
241      status = nxd_ipv6_global_address_set(&client_ip, &client_ip_address, 64);
242  #endif  /* MULTIHOME_NETXDUO */
243
244      /* Check for global address set error.  */
245      if (status != NX_SUCCESS)
246      {
247          return;
248      }
249
250      /* Wait while NetX Duo validates the global and link local addresses. */
251      tx_thread_sleep(400);
252
253  #endif
254
255
256      /* Set up client time updates depending on mode. */
257  #ifdef USE_UNICAST
258
259          /* Initialize the Client for unicast mode to poll the SNTP server once an
hour. */
260  #ifdef USE_IPV6
261          /* Use the duo service to set up the Client and set the IPv6 SNTP server.
Note: this
262             can take either an IPv4 or IPv6 address. */
263          status = nxd_sntp_client_initialize_unicast(&demo_client,
                                                        &sntp_server_address);
264  #else
265          /* Use the IPv4 service to set up the Client and set the IPv4 SNTP server.
*/
266          status = nx_sntp_client_initialize_unicast(&demo_client,
                                                                SERVER_IP_ADDRESS);
267  #endif  /* USE_IPV6 */
268
269
270  #else   /* Broadcast mode */
```

```
271
272          /* Initialize the Client for broadcast mode, no roundtrip calculation
                required and a broadcast SNTP service. */
273
274  #ifdef USE_IPV6
275          /* Use the duo service to initialize the Client and set IPv6 SNTP all hosts
              multicast address.
276             (Note: This can take either an IPv4 or IPv6 address.)*/
277          status = nxd_sntp_client_initialize_broadcast(&demo_client,
                                                  &sntp_server_address, NX_NULL);
278  #else
279
280          /* Use the IPv4 service to initialize the Client and set IPv4 SNTP
              broadcast address. */
281          status = nx_sntp_client_initialize_broadcast(&demo_client,  NX_NULL,
SERVER_IP_ADDRESS);
282  #endif  /* USE_IPV6 */
283  #endif  /* USE_UNICAST */
284
285
286      /* Check for error. */
287      if (status != NX_SUCCESS)
288      {
289
290          return;
291      }
292
293      /* Set the base time which is approximately the number of seconds since the
           turn of the last century. If this is not available in SNTP format, the
           nx_sntp_client_utility_add_msecs_to_ntp_time service can convert
           milliseconds to fraction.  For how to compute NTP seconds from real time,
           read the NetX Duo SNTP User Guide.

          Otherwise set the base time to zero and set
           NX_SNTP_CLIENT_IGNORE_MAX_ADJUST_STARTUP to NX_TRUE for
          the SNTP CLient to accept the first time update without applying a minimum
          or maximum adjustment  parameters (NX_SNTP_CLIENT_MIN_TIME_ADJUSTMENT and
          NX_SNTP_CLIENT_MAX_TIME_ADJUSTMENT). */
301
302      base_seconds =  0xd2c96b90;  /* Jan 24, 2012 UTC */
303      base_fraction = 0xa132db1e;
304
305      /* Apply to the SNTP Client local time.  */
306      status = nx_sntp_client_set_local_time(&demo_client, base_seconds,
                                                base_fraction);
307
308      /* Check for error. */
309      if (status != NX_SUCCESS)
310      {
311
312          return;
313      }
314
315      /* Run whichever service the client is configured for. */
316  #ifdef USE_UNICAST
317
318      status = nx_sntp_client_run_unicast(&demo_client);
319  #else
320      status = nx_sntp_client_run_broadcast(&demo_client);
321  #endif  /* USE_UNICAST */
322
323      if (status != NX_SUCCESS)
324      {
325          return;
326      }
327
328      spin = NX_TRUE;
329
330      /* Now check periodically for time changes. */
331      while(spin)
332      {
```

```
333
334         /* First verify we have a valid SNTP service running. */
335         status = nx_sntp_client_receiving_updates(&demo_client, &server_status);
336
337         if ((status == NX_SUCCESS) && (server_status == NX_TRUE))
338         {
339
340             /* Server status is good. Now get the Client local time. */
341
342             /* Display the local time in years, months, date format.  */
343             status = nx_sntp_client_get_local_time(&demo_client, &seconds,
                                                       &milliseconds, &buffer[0]);
344
345             if (status == NX_SUCCESS)
346             {
347                 printf("Date: %s\n", &buffer[0]);
348             }
349
350             /* Wait a while before the next update. */
351             tx_thread_sleep(300);
352
353             memset(&buffer[0], 0, BUFSIZE);
354         }
355         else
356         {
357
358             /* Wait a short bit to check again. */
359             tx_thread_sleep(100);
360         }
361     }
362
363     /* We can stop the SNTP Client if for example the SNTP server has stopped. */
364     status = nx_sntp_client_stop(&demo_client);
365
366     if (status != NX_SUCCESS)
367     {
368         return;;
369     }
370
371     /* Set up another server and reinitialize the SNTP Client. */
372 #ifdef USE_UNICAST
373 #ifdef USE_IPV6
374
375     status = nxd_sntp_client_initialize_unicast(&demo_client,
&sntp_server_address);
376
377 #else
378     /* Initialize the Client for unicast mode to poll the SNTP server once an hour.
*/
379     status = nx_sntp_client_initialize_unicast(&demo_client, SERVER_IP_ADDRESS_2);
380 #endif
381
382     /* Check for error. */
383     if (status != NX_SUCCESS)
384     {
385         return;
386     }
387
388     /* Now start the SNTP Client task back up. */
389     status = nx_sntp_client_run_unicast(&demo_client);
390
391     if (status != NX_SUCCESS)
392     {
393         return;
394
395     }
396 #else    /* Start Client in broadcast */
397
398     /* Initialize the Client for broadcast mode (multicast in IPv6) and set up an
        alternative server. */
399 #ifdef USE_IPV6
```

```
400
401     status = nxd_sntp_client_initialize_broadcast(&demo_client,
                                                      &sntp_server_address2, NX_NULL);
402 #else
403
404     status = nx_sntp_client_initialize_broadcast(&demo_client, NX_NULL,
                                                                   SERVER_IP_ADDRESS_2)
                                                                  ;
405 #endif /* USE_IPV6*/
406
407     if (status != NX_SUCCESS)
408     {
409         return;
410     }
411
412     /* Now start the SNTP Client task back up. */
413     status = nx_sntp_client_run_broadcast(&demo_client);
414
415     /* Check for error. */
416     if (status != NX_SUCCESS)
417     {
418         return;
419     }
420 #endif
421
422     spin = NX_TRUE;
423
424     /* Now check periodically for time changes. */
425     while(spin)
426     {
427
428         /* First verify we have a valid SNTP service running. */
429         status = nx_sntp_client_receiving_updates(&demo_client, &server_status);
430
431         if ((status == NX_SUCCESS) && (server_status == NX_TRUE))
432         {
433
434             /* Server status is good. Now retrieve the Client local time. */
435
436             /* Display the local time in years, months, date format.  */
437             status = nx_sntp_client_get_local_time(&demo_client, &seconds,
                                                       &milliseconds, &buffer[0]);
438             if (status == NX_SUCCESS)
439             {
440                 printf("Date: %s\n", &buffer[0]);
441             }
442
443             /* It will be a bit longer till the next update. */
444             tx_thread_sleep(200);
445
446             memset(&buffer[0], 0, BUFSIZE);
447         }
448
449         /* Wait a short bit and try again. */
450         tx_thread_sleep(100);
451     }
452
453     /* To return resources to the system, delete the SNTP. */
454     status = nx_sntp_client_delete(&demo_client);
455
456     return;
457 }
458
459
460 /* This application defined handler for handling an impending leap second is not
461    required by the SNTP Client. The default handler below only logs the event for
462    every time stamp received with the leap indicator set.  */
463
464 UINT leap_second_handler(NX_SNTP_CLIENT *client_ptr, UINT leap_indicator)
465 {
466
```

```
467      /* Handle the leap second handler... */
468
469      return NX_SUCCESS;
470  }
471
472  /* This application defined handler for handling a Kiss of Death packet is not
473     required by the SNTP Client. A KOD handler should determine
474     if the Client task should continue vs. abort sending/receiving time data
475     from its current time server, and if aborting if it should remove
476     the server from its active server list.
477
478     Note that the KOD list of codes is subject to change. The list
479     below is current at the time of this software release. */
480
481  UINT kiss_of_death_handler(NX_SNTP_CLIENT *client_ptr, UINT KOD_code)
482  {
483
484  UINT    remove_server_from_list = NX_FALSE;
485  UINT    status = NX_SUCCESS;
486
487
488      /* Handle kiss of death by code group. */
489      switch (KOD_code)
490      {
491
492          case NX_SNTP_KOD_RATE:
493          case NX_SNTP_KOD_NOT_INIT:
494          case NX_SNTP_KOD_STEP:
495
496              /* Find another server while this one is temporarily out of service.
*/
497              status =  NX_SNTP_KOD_SERVER_NOT_AVAILABLE;
498
499          break;
500
501          case NX_SNTP_KOD_AUTH_FAIL:
502          case NX_SNTP_KOD_NO_KEY:
503          case NX_SNTP_KOD_CRYP_FAIL:
504
505              /* These indicate the server will not service client with time updates
506                 without successful authentication. */
507
508
509              remove_server_from_list =  NX_TRUE;
510
511          break;
512
513
514          default:
515
516              /* All other codes. Remove server before resuming time updates. */
517
518              remove_server_from_list =  NX_TRUE;
519          break;
520      }
521
522      /* Removing the server from the active server list? */
523      if (remove_server_from_list)
524      {
525
526          /* Let caller know it switch SNTP servers before resuming SNTP Client. */
527          status = NX_SNTP_KOD_REMOVE_SERVER;
528      }
529
530      return status;
531  }
532
```

Figure 1 Example of using SNTP Client with NetX Duo

Below in Figure 2 is a modification of the example shown in Figure 1 above to demonstrate how to use the multi home interface feature of NetX Duo. Inserted below line 45-49 where NetX Duo and ThreadX resource variables are created, the host's primary and secondary interface IP addresses are defined, as well as the host IP gateway address (optional) and server list of time servers. Notice that these are real IP addresses, and not the simulator IP addresses used by for the NetX ram driver demo, for purposes of demonstration.

After the Client IP instance is created in lines 94-104 using the primary client interface address, the second host interface is 'attached' to the main IP control block with the secondary address and in this case the same network driver in lines 112-119.

Lastly, once the client thread is running, the Client IP gateway is set at the top of the *demo_client_thread_entry* function in lines 200-204. This last step is **only** necessary if any of the host's time servers are located on an off link network address and all packets must go through the host gateway to reach them.

At this point the IP task will be able to figure out which interface to send out packets to regardless if the host client connects to its time server through the primary or secondary interface. See the NetX User Guide for more specific information *on nx_ip_interface_attach* and *nx_ip_gateway_address_set.*

```
43   /* Set up client thread and network resources. */
44
45   NX_PACKET_POOL      client_packet_pool;
46   NX_IP               client_ip;
47   NX_UDP_SOCKET       client_socket;
48   TX_THREAD           demo_client_thread;
49   NX_SNTP_CLIENT      demo_client;
50
51   #define SERVER_IP_ADDRESS        "64.125.78.85  192.2.2.92"
52   #define CLIENT_PRIMARY_ADDRESS    IP_ADDRESS(192,68,1,10)
53   #define CLIENT_SECONDARY_ADDRESS IP_ADDRESS( 64,125,78,85)
54   #define GATEWAY_IP_ADDRESS       IP_ADDRESS(192,68,1,1)
55
56   #define MULTI_HOMED_DEVICE        1
```

…

```
93       /* Create Client IP instances */
94       status = nx_ip_create(&client_ip, "SNTP IP Instance", NX_SNTP_CLIENT_IP_ADDRESS,
95                             0xFFFFFF00UL, &client_packet_pool,  nx_etherDriver_mcf5272,
96                             free_memory_pointer, NX_SNTP_CLIENT_IP_STACK_SIZE,
97                             NX_SNTP_CLIENT_IP_THREAD_PRIORITY);
98
99       /* Check for error. */
100      if (status != NX_SUCCESS)
101      {
102
103          NX_SNTP_CLIENT_EVENT_LOG(SEVERE, ("Error creating IP instance. Status: 0x%x\n\r", status));
105          return;
106      }
107
108      free_memory_pointer =  free_memory_pointer + NX_SNTP_CLIENT_IP_STACK_SIZE;
109
110      #ifdef MULTI_HOMED_DEVICE
111      /* Create the second Client Interface.  */
112      status =  _nx_ip_interface_attach(&client_ip, "port_2", CLIENT_SECONDARY_ADDRESS,
113                  0xFFFFFF00UL, nx_etherDriver_mcf5485);
114
```

```
115      /* Check for IP attach errors.  */
116      if (status)
117      {
118          return;
119      }
120      #endif
...
188  /* Define the client thread.  */
189  void    demo_client_thread_entry(ULONG info)
190  {
191
192  UINT                status;
193  NX_SNTP_CLIENT      *client_ptr;
194
195
196      client_ptr = (NX_SNTP_CLIENT *)info;
197
198
199    /* For each off link SNTP server IP address, a next hop (e.g. gateway) must be established. */
200    status = nx_ip_gateway_address_set(client_ptr -> ip_ptr, GATEWAY_IP_ADDRESS);
201    if (status)
202    {
203
204        return;
205    }
...
```

Figure 2 Example of using a multi homed SNTP Client host with NetX (5.3 or later)

Configuration Options

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

Define	Meaning
NX_DISABLE_ERROR_CHECKING	Defined, this option removes the basic SNTP error checking. It is typically used after the application has been debugged.
NX_SNTP_CLIENT_THREAD_STACK_SIZE	This option sets the size of the Client thread stack. The default NetX Duo 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 Duo SNTP Client size is TX_NO_TIME_SLICE.
NX_SNTP_CLIENT_ THREAD_PRIORITY	This option sets the Client thread priority. The NetX Duo 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 Duo 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 Duo 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 Duo 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 Duo 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 Duo SNTP Client socket. Additional packets received mean the oldest packets are released. The default NetX Duo SNTP Client is set to 5.

NX_SNTP_CLIENT_PACKET_SIZE Size of the UDP packet for sending time requests out. This includes UDP, IP, and Ethernet (Frame) packet header data. The default NetX Duo SNTP Client is 122 bytes.

NX_SNTP_CLIENT_PACKET_POOL_SIZE
Size of the SNTP Client packet pool. The NetX Duo SNTP Client default is (10 * NX_SNTP_CLIENT_PACKET_SIZE).

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

NX_SNTP_CLIENT_NTP_VERSION SNTP version used by the Client The NetX Duo SNTP Client API was based on Version 4.

NX_SNTP_CLIENT_MIN_NTP_VERSION Oldest SNTP version the Client will be able to work with. The NetX Duo 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 Duo 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 Duo 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 Duo 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 Duo SNTP Client default is enabled.

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 10.

NX_SNTP_CLIENT_UNICAST_POLL_INTERVAL

The starting poll interval (seconds) on which the Client in unicast mode sends a time request to its SNTP server. The NetX Duo 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_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 zero. The NetX Duo SNTP Client default is set to 500.

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 Duo 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.

NX_SNTP_CLIENT_SLEEP_INTERVAL The time interval during which the SNTP Client task sleeps. This allows the host application API calls to be executed by the SNTP Client. The default value is 1 timer tick.

NX_SNTP_CURRENT_YEAR

This should be set to the number of seconds from 1900 Jan 1 to the current year for the SNTP Client to be able to display the local time in years, month and date. The default value is zero.

Chapter 3

Description of NetX Duo SNTP Client Services

This chapter contains a description of all NetX Duo 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 IPv4 broadcast operation

nxd_sntp_client_initialize_broadcast
Initialize Client for IPv6 or IPv4 broadcast operation

nx_sntp_client_initialize_unicast
Initialize Client for IPv4 unicast operation

nxd_sntp_client_initialize_unicast
Initialize Client for IPv4 or IPv6 unicast operation

nx_sntp_client_receiving_udpates
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_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

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

Return Values

NX_SUCCESS	(0x00)	Successful Client creation
NX_SNTP_INSUFFICIENT_PACKET_PAYLOAD	(0xD2A)	Invalid non pointer input
NX_PTR_ERROR	(0x16)	Invalid pointer input
NX_INVALID_PARAMETERS	(0x16)	Invalid non pointer input

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.  */
```

See Also

nx_sntp_client_delete

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

client_ptr	Pointer to SNTP Client control block

Return Values

NX_SUCCESS	(0x00)	Successful Client creation
NX_PTR_ERROR	(0x16)	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.  */
```

See Also

nx_sntp_client_create

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

client_ptr	Pointer to SNTP Client control block
seconds	Pointer to local time seconds
milliseconds	Pointer to milliseconds component
buffer	Pointer to buffer to write time data

Return Values

NX_SUCCESS	(0x00) Successful Client creation
NX_PTR_ERROR	(0x16) 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.  */
```

See Also

nx_sntp_client_set_local_time

nx_sntp_client_initialize_broadcast

Initialize the Client for broadcast operation

Prototype

```
UINT nx_sntp_client_initialize_broadcast(NX_SNTP_CLIENT *client_ptr,
            ULONG multicast_server_address,
            ULONG broadcast_time_servers);
```

Description

This service initializes the Client for broadcast operation by setting up the SNTP Server IP address and initializing SNTP startup parameters and timeouts. If both multicast and broadcast addresses are non null, the multicast address is selected. If both addresses are null an error is returned

Input Parameters

client_ptr	Pointer to SNTP Client control block
multicast_server_address	SNTP multicast address
broadcast_time_server	SNTP server broadcast address

Return Values

NX_SUCCESS	(0x00)	Client successfully initialized
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Initialization, Threads

Example

```
/* Initialize the client for broadcast operation.  */
status =  nx_sntp_client_initialize_broadcast(client_ptr,0x0,
                                    NX_NULL, IP_ADDRESS(192,2,2,255);

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

See Also

nxd_sntp_client_initialize_broadcast, nx_sntp_client_run_broadcast, nx_sntp_client_initialize_unicast, nx_sntp_client_run_unicast

nxd_sntp_client_initialize_broadcast

Initialize the Client for IPv4 or IPv6 broadcast operation

Prototype

```
UINT nxd_sntp_client_initialize_broadcast(NX_SNTP_CLIENT *client_ptr,
            NXD_ADDRESS *multicast_server_address,
            NXD_ADDRESS *broadcast_server_address);
```

Description

This service initializes the Client for broadcast operation by setting up the SNTP Server IP address and initializing SNTP startup parameters and timeouts. If both broadcast and multicast address pointers are non null, the multicast address is selected. If both address pointers are null, an error is returned. This supports both IPv4 and IPv6 address types. Note that IPv6 does not support broadcast, so the broadcast address pointer is set to IPv6, an error is returned.

Input Parameters

client_ptr	Pointer to SNTP Client control block
multicast_server_address	SNTP server multicast address
broadcast_server_address	SNTP server broadcast address

Return Values

NX_SUCCESS	(0x00)	Client successfully initialized
NX_SNTP_PARAM_ERROR	(0xD0D)	Invalid non pointer input
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Initialization**,** Threads

Example

```
/* Initialize the client for broadcast operation.  */
NXD_ADDRESS broadcast_server;

Broadcast_server.nxd_ip_address = NX_IP_VERSION_V6;
Broadcast_server.nxd_ip_address.v6[0] = 0x20010db1;
Broadcast_server.nxd_ip_address.v6[1] = 0x0f101;
Broadcast_server.nxd_ip_address.v6[2] = 0x0;
Broadcast_server.nxd_ip_address.v6[3] = 0x101;

status =  nxd_sntp_client_initialize_broadcast(client_ptr,0x0,
                                    NX_NULL, &broadcast_server)


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

See Also

Nxd_sntp_client_initialize_broadcast, nx_sntp_client_run_broadcast, nx_sntp_client_initialize_unicast, nx_sntp_client_run_unicast

nx_sntp_client_initialize_unicast

Set up the SNTP Client to run in unicast

Prototype

```
UINT nx_sntp_client_initialize_unicast(NX_SNTP_CLIENT * client_ptr,
                                       ULONG unicast_time_server);
```

Description

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

Input Parameters

client_ptr	Pointer to SNTP Client control block
unicast_time_server	SNTP server IP address

Return Values

NX_SUCCESS	(0x00)	Client successfully initialized
NX_INVALID_PARAMETERS	(0x40)	Invalid non pointer input
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Initialization, Threads

Example

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

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

See Also

nx_sntp_client_initialize_unicast, nx_sntp_client_run_unicast, nx_sntp_client_run_broadcast

nxd_sntp_client_initialize_unicast

Set up the SNTP Client to run in IPv4 or IPv6 unicast

Prototype

```
UINT nxd_sntp_client_initialize_unicast(NX_SNTP_CLIENT * client_ptr,
                                        NXD_ADDRESS *unicast_time_server);
```

Description

This service initializes the Client for unicast operation by setting up the SNTP Server IP address and initializing SNTP startup parameters and timeouts. This supports both IPv4 and IPv6 address types.

Input Parameters

client_ptr	Pointer to SNTP Client control block
unicast_time_server	SNTP server IP address

Return Values

NX_SUCCESS	(0x00)	Client successfully initialized
NX_INVALID_PARAMETERS	(0x40)	Invalid non pointer input
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Initialization, Threads

Example

```
/* Initialize the Client for unicast operation.  */
NXD_ADDRESS unicast_server;

unicast _server.nxd_ip_address = NX_IP_VERSION_V6;
unicast _server.nxd_ip_address.v6[0] = 0x20010db1;
unicast _server.nxd_ip_address.v6[1] = 0x0f101;
unicast _server.nxd_ip_address.v6[2] = 0x0;
unicast _server.nxd_ip_address.v6[3] = 0x101;

status =  nxd_sntp_client_initialize_unicast(&client_ptr, *unicast_server);
```

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

See Also

nxd_sntp_client_initialize_unicast, nx_sntp_client_run_unicast, nx_sntp_client_run_broadcast

nx_sntp_client_receiving_updates

Return status if Client 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 host 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

client_ptr	Pointer to SNTP Client control block.
receive_status	Pointer to indicator if Client is receiving valid updates.

Return Values

NX_SUCCESS	(0x00)	Client successfully initialized
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Initialization, Threads

Example

```
/* Start Client running in broadcast mode.  */
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.  */
```

See Also

nx_sntp_client_initialize_broadcast, nx_sntp_client_initialize_unicast, nx_sntp_client_run_unicast

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 the either of these limits are exceeded, the SNTP Client sets the server status to invalid although it will still wait to receive updates. The host 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

client_ptr	Pointer to SNTP Client control block.

Return Values

NX_SUCCESS	(0x00) Client successfully initialized
NX_SNTP_CLIENT_ALREADY_STARTED	(0xD0C) Client already started
NX_SNTP_CLIENT_NOT_INITIALIZED	(0xD01) Client not initialized
NX_PTR_ERROR	(0x16) Invalid pointer input

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.  */
```

See Also

nx_sntp_client_initialize_broadcast, nx_sntp_client_initialize_unicast, nx_sntp_client_run_unicast

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 will wait to receive broadcasts from the SNTP server. 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 host 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

client_ptr	Pointer to SNTP Client control block.

Return Values

NX_SUCCESS	(0x00)	Client successfully initialized
NX_SNTP_CLIENT_ALREADY_STARTED	(0xD0C)	Client already started
NX_SNTP_CLIENT_NOT_INITIALIZED	(0xD01)	Client not initialized
NX_PTR_ERROR	(0x16)	Invalid pointer input

Allowed From

Threads

Example

```
/* Start the Client in unicast mode. */
status =  nx_sntp_client_run_unicast(client_ptr);

/* If status = NX_SUCCESS, the Client was successfully started.  */
```

See Also

nx_sntp_client_initialize_unicastnx_sntp_client_initialize_broadcast, nx_sntp_client_run_broadcast, nx_sntp_client_send_unicast_request,

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

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

Return Values

NX_SUCCESS	(0x00)	Successful Client creation
NX_SNTP_INVALID_TIME	(0xD30)	Successful Client creation
NX_PTR_ERROR	(0x16)	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.  */
```

See Also

nx_sntp_client_get_local_time

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 host applications to make API calls on the SNTP Client.

Input Parameters

client_ptr	Pointer to SNTP Client control block

Return Values

NX_SUCCESS	(0x00)	Successful Client creation
NX_PTR_ERROR	(0x16)	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.  */
```

See Also

nx_sntp_client_initialize_broadcast, nx_sntp_client_ initialize _unicast, nxd_sntp_client_initialize_broadcast, nxd_sntp_client_initialize_unicast, nx_sntp_client_run_broadcast, nx_sntp_client_run_unicast

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. It requires that the NX_SNTP_CURRENT_YEAR be configured e.g. usually the current year.

Input Parameters

client_ptr	Pointer to SNTP Client
buffer	Pointer to buffer to store date
length	Size of input buffer

Return Values

NX_SUCCESS	(0x00)	Successful conversion
NX_SNTP_ERROR_CONVERTING_DATETIME	(0xD08)	Error converting time to a date
NX_SNTP_INVALID_DATETIME_BUFFER	(0xD07)	Insufficient buffer length

Allowed From

Initialization, Threads

Example

```
/* Display the Client's local time. */
status =  nx_sntp_client_utility_display_date_time (client_ptr,
              NX_SNTP_CURRENT_YEAR, current_date_time_ptr);
```

```
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

`milliseconds`	Milliseconds to convert
`fraction`	Pointer to millseconds converted to fraction

Return Values

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

Allowed From

Initialization, Threads

Example

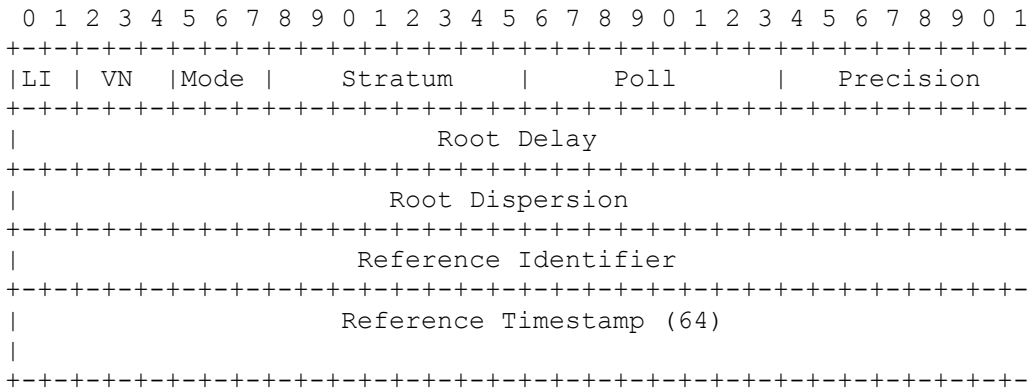
```
/* Convert the milliseconds to a fraction. */

status =  nx_sntp_client_utility_msecs_to_fraction(milliseconds, &fraction);
/* If status is NX_SUCCESS, data was successfully converted.  */
```

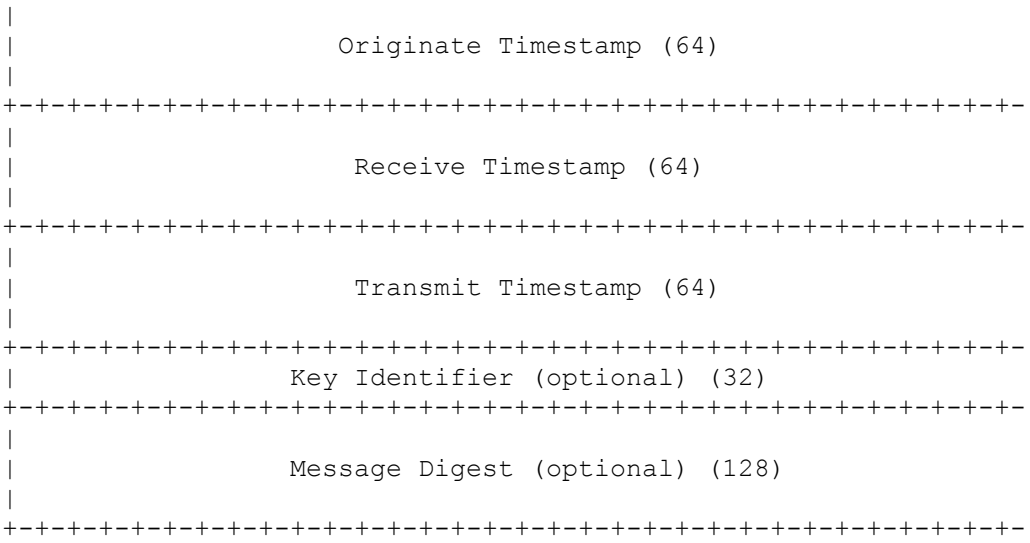
Appendix A NTP Time Stamp Format

The SNTP protocol uses the identical time stamp format, shown in Figure 2 below, as NTP, as part of the effort to keep NTP and SNTP hosts interoperable. The NTP time stamp contains several descriptor fields mostly for the server to share information about its time service, such as NTP version, clock precision, mode of operation, leap second warning, and NTP stratum. The remaining required fields are time stamp fields for recording when a request was received, when it was transmitted and when the server clock was last synchronized. The time stamp also contains two optional fields for authentication, Key Identifier and Message Digest.

Each time stamp represents time in a 64 bit field. The upper 32 bits contain time since the turn of the previous century (01-01-1900) in seconds, and the lower 32 bits contain the fraction of a second in fixed point notation. The SNTP Client API contains the tables and conversion formulas used in the Network Time Protocol Distribution Version 4 software (http://www.ntp.org/downloads.html) for converting time fractions in to milliseconds and microseconds. Using this format, the NTP time format will run out of range in a 32 bit field in the year 2032. The proposed plan is to roll over the seconds using an as yet unimplemented 'epoch' field which will be incremented by one for each block of time where the seconds must be rolled over. For an extensive discussion on this topic, visit http://www.eecis.udel.edu/~mills/y2k.html.

Note: The NetX Duo SNTP Client API includes a utility for displaying NTP time in human readable format using the *nx_sntp_client_utility_convert_seconds_to_date()* and *_nx_sntp_client_utility_display_date_time()* function calls. This is demonstrated in the demo program in the *Examples* directory of the NetX Duo SNTP Client package.





NTP time stamp format

For SNTP client messages, most of these fields are zero.

Leap Indicator (LI): This is a two-bit code warning of an impending leap second to be inserted or deleted in the last minute of the current day. This field is significant only in server messages, where the values are defined as follows:

LI	Meaning
0	No warning
1	Last minute has 61 seconds
2	Last minute has 59 seconds
3	Alarm condition (clock not synchronized)

On startup, servers set this field to 3 until the server clock is synchronized.

Version Number (VN): This is a three-bit integer indicating the NTP or SNTP version number.

Mode: This is a three-bit number indicating the protocol mode. The values are defined as follows:

Mode	Meaning
0	Reserved
1	Symmetric active
2	Symmetric passive
3	Client

4	Server
5	Broadcast (server)
6	Reserved for NTP control message
7	Reserved for private use

In unicast and manycast modes, the client sets this field to 3 (client) in the request, and the server sets it to 4 (server) in the reply. In broadcast mode, the server sets this field to 5 (broadcast). The other modes are not used by SNTP servers and clients.

Stratum**:** This is an eight-bit unsigned integer indicating the Stratum or hierarchy among time servers. A '1' indicates the top level of the hierarchy (server) and anything lower, down to 15, is either a server or more likely a client. This field is significant only in SNTP server messages. Its values are defined as follows:

Stratum	Meaning
0	kiss-o'-death message (see below)
1	primary reference (e.g., synchronized by radio clock)
2-15	secondary reference (synchronized by NTP or SNTP)
16-255	reserved

Poll Interval**:** This is an eight-bit unsigned integer used as an exponent of two, where the resulting value is the maximum interval between successive messages in seconds. This field is significant only in SNTP server messages, where the values range from 4 (16 s) to 17 (131,072 s -- about 36 h).

Precision: This is an eight-bit signed integer used as an exponent of two, where the resulting value is the precision of the system clock in seconds. This field is significant only in server messages, where the values range from -6 for mains-frequency clocks to -20 for microsecond clocks found in some workstations.

Root Delay: This is a 32-bit signed fixed-point number indicating the total roundtrip delay to the primary reference source, in seconds with the fraction point between bits 15 and 16. This data is not used in the SNTP Client API.

Reference Identifier Codes:

Code	External Reference Source
LOCL	Uncalibrated local clock
CESM	Calibrated Cesium clock

RBDM	Calibrated Rubidium clock
PPS	Calibrated quartz clock or other pulse-per-second source
IRIG	Inter-Range Instrumentation Group
ACTS	NIST telephone modem service
USNO	USNO telephone modem service
PTB	PTB (Germany) telephone modem service
TDF	Allouis (France) Radio 164 kHz
DCF	Mainflingen (Germany) Radio 77.5 kHz
MSF	Rugby (UK) Radio 60 kHz
WWV	Ft. Collins (US) Radio 2.5, 5, 10, 15, 20 MHz
WWVB	Boulder (US) Radio 60 kHz
WWVH	Kauai Hawaii (US) Radio 2.5, 5, 10, 15 MHz
CHU	Ottawa (Canada) Radio 3330, 7335, 14670 kHz
LORC	LORAN-C radio navigation system
OMEG	OMEGA radio navigation system
GPS	Global Positioning Service

Root Dispersion**:** This is a 32-bit unsigned fixed-point number indicating the maximum error in the server clock, in seconds with the fraction point between bits 15 and 16. This field is significant only in server messages, where the values range from zero to several hundred microseconds.

Reference Identifier**:** This is a 32-bit bit string identifying the particular reference source. This field is significant only in server messages, where for stratum 0 (kiss-o'-death message) and 1 (primary server), the value is a four-character ASCII string, left justified and zero padded to 32 bits. Primary (stratum 1) servers set their Reference Identifier to a code identifying the external reference source according to Figure 3 above. If the external reference is one of those listed, the associated code should be used.

Reference Timestamp: This field is the time the system clock was last set or corrected, in 64-bit timestamp format.

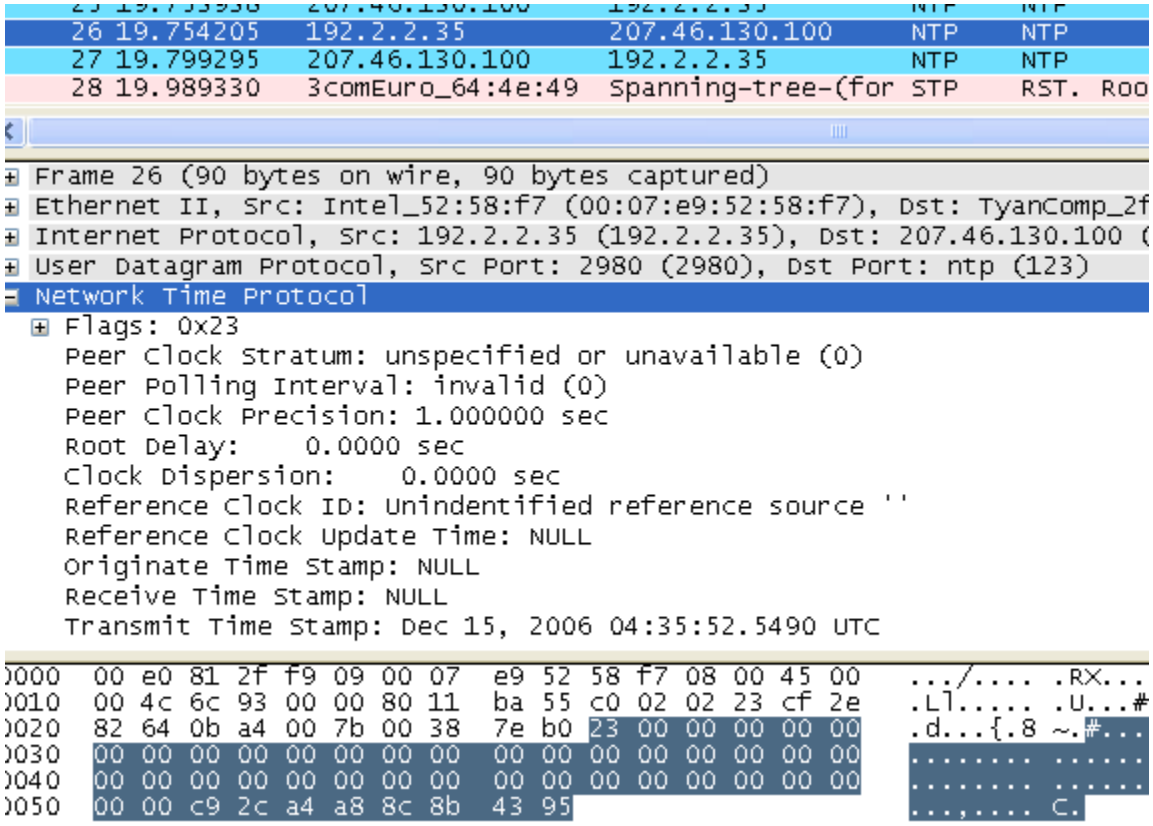
Originate Timestamp: This is the time at which the request departed the client for the server, in 64-bit timestamp format.

Receive Timestamp: This is the time at which the request arrived at the server or the reply arrived at the client, in 64-bit timestamp format.

Transmit Timestamp: This is the time at which the request departed the client or the reply departed the server, in 64-bit timestamp format.

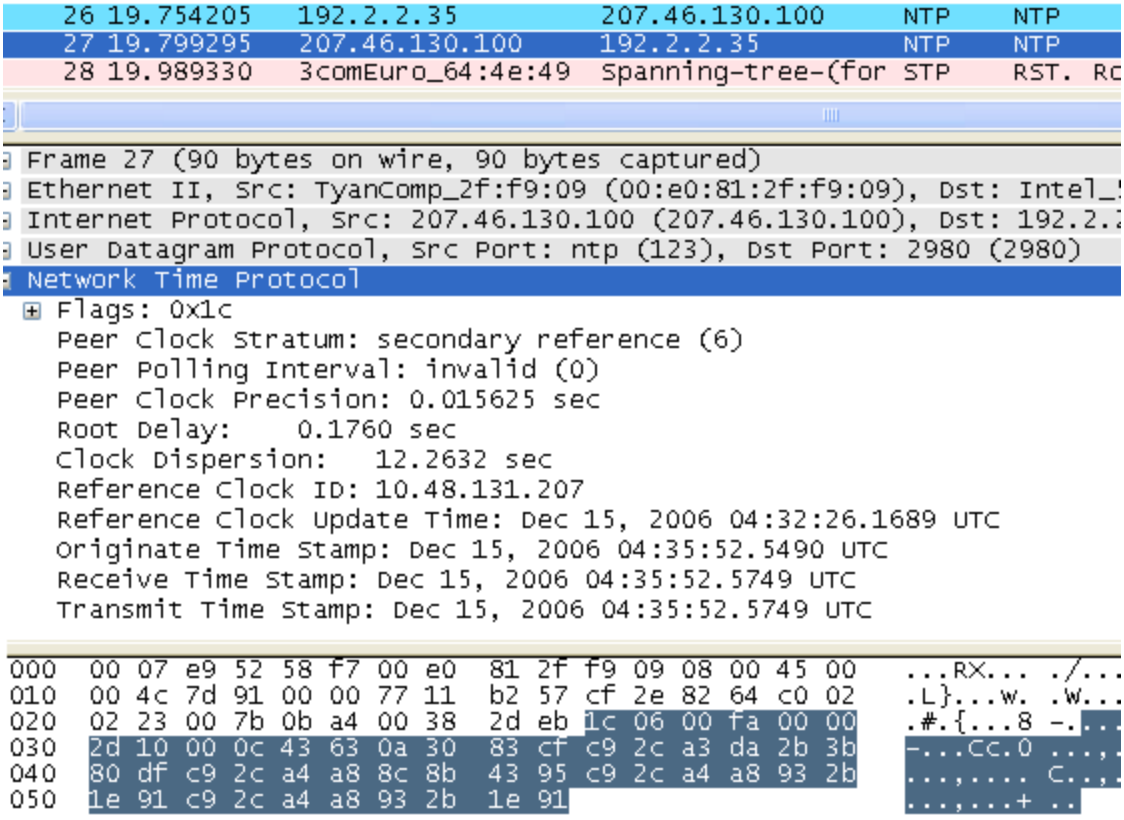
Authenticator (optional): When the NTP authentication scheme is implemented, the Key Identifier and Message Digest fields contain the message authentication code (MAC) information defined in Appendix C of RFC 1305.

Below is an actual unicast request (poll) to 207.46.130.100, a stratum 1 server from the MCF 5272 processor 192.2.2.35:



Unicast Client poll request

Below is an actual unicast reply from 207.46.130.100, a stratum 1 server, responding to a unicast poll from the MCF 5272 processor 192.2.2.35:



Unicast Server reply

Appendix B 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 host 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.h.* The API to manipulate the SNTP Client list is shown below. More information is available in **Chapter 4 Description of SNTP Client Services.**

_nx_sntp_client_get_next_server
_nx_sntp_client_remove_server_from_list

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

NX_SNTP_KOD_REMOVE_SERVER	0xD0C
NX_SNTP_SERVER_AUTH_FAIL	0xD0D
NX_SNTP_INVALID_NTP_VERSION	0xD11
NX_SNTP_INVALID_SERVER_MODE	0xD12
NX_SNTP_INVALID_SERVER_STRATUM	0xD15

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

NX_SNTP_NO_UNICAST_FROM_SERVER	0xD09
NX_SNTP_SERVER_CLOCK_NOT_SYNC	0xD0A
NX_SNTP_KOD_SERVER_NOT_AVAILABLE	0xD0B
NX_SNTP_OVER_BAD_UPDATE_LIMIT	0xD17
NX_SNTP_BAD_SERVER_ROOT_DISPERSION	0xD16
NX_SNTP_INVALID_RTT_TIME	0xD21
NX_SNTP_KOD_SERVER_NOT_AVAILABLE	0xD24