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

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 5.10

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 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 manycast.

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:

nx_sntp_client.c	SNTP Client C source file
nx_sntp_client.h	SNTP Client Header file
demo_netx_sntp_client.c	Demonstration SNTP Client application
nx_sntp.docx	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
```

```
   Protocol in unicast and broadcast modes.

 */


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

/* Set up generic network driver for demo program. */
void    _nx_ram_network_driver(struct NX_IP_DRIVER_STRUCT *driver_req);

/* 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 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;
UCHAR    *free_memory_pointer;


    free_memory_pointer = (UCHAR *)first_unused_memory;

    /* Create client packet pool. */
    status =  nx_packet_pool_create(&client_packet_pool, "SNTP Client Packet Pool",
                      NX_SNTP_CLIENT_PACKET_SIZE, free_memory_pointer,
                      NX_SNTP_CLIENT_PACKET_POOL_SIZE);

    /* Check for errors. */
    if (status != NX_SUCCESS)
    {

        return;
    }

    /* Initialize the NetX system. */
    nx_system_initialize();

    /* Update pointer to unallocated (free) memory. */
    free_memory_pointer =  free_memory_pointer + NX_SNTP_CLIENT_PACKET_POOL_SIZE;

    /* Create Client IP instances */
    status = nx_ip_create(&client_ip, "SNTP IP Instance", CLIENT_IP_ADDRESS,
                0xFFFFFF00UL, &client_packet_pool, _nx_ram_network_driver,
                free_memory_pointer, 2048, 1);

    /* Check for error. */
    if (status != NX_SUCCESS)
    {

        return;
    }

    free_memory_pointer =  free_memory_pointer + 2048;

    /* Enable ARP and supply ARP cache memory. */
    status =  nx_arp_enable(&client_ip, (void **) free_memory_pointer, 2048);

    /* Check for error. */
    if (status != NX_SUCCESS)
    {

        return;
    }

    /* Update pointer to unallocated (free) memory. */
    free_memory_pointer = free_memory_pointer + 2048;

    /* Enable UDP for client. */
    status =  nx_udp_enable(&client_ip);

    /* Check for error. */
    if (status != NX_SUCCESS)
    {

        return;
    }

    status = nx_icmp_enable(&client_ip);

    /* Check for error. */
    if (status != NX_SUCCESS)
    {

        return;
    }
    /* Create the client thread */
    status = tx_thread_create(&demo_client_thread, "SNTP Client Thread", demo_client_thread_entry,
                              (ULONG)(&demo_client), free_memory_pointer, 2048,
                              4, 4, TX_NO_TIME_SLICE, TX_DONT_START);

    /* Check for errors */
    if (status != TX_SUCCESS)
    {

        return;
    }

    /* Update pointer to unallocated (free) memory. */
    free_memory_pointer = free_memory_pointer + 2048;

    /* set the SNTP network interface to the primary interface. */
    iface_index = 0;

    /* Create the SNTP Client to run in broadcast mode.. */
    status =  nx_sntp_client_create(&demo_client, &client_ip, iface_index, &client_packet_pool,
                                    leap_second_handler,
                                    kiss_of_death_handler,
                                    NULL /* no random_number_generator callback */);

    /* Check for error. */
    if (status != NX_SUCCESS)
    {

        /* Bail out!*/
        return;
    }

    tx_thread_resume(&demo_client_thread);

    return;
}

/* Define size of buffer to display client's local time. */
#define BUFSIZE 50

/* Define the client thread.  */
void    demo_client_thread_entry(ULONG info)
{
```

```
UINT   status;
UINT   spin;
UINT   server_status;
CHAR   buffer[BUFSIZE];
ULONG  base_seconds;
ULONG  base_fraction;
ULONG  seconds, milliseconds;


    /* Give other threads (IP instance) a chance to initialize. */
    tx_thread_sleep(100);



    /* Set up client time updates depending on mode. */
#ifdef USE_UNICAST

    /* Initialize the Client for unicast mode to poll the SNTP server once an hour. */
    /* Use the IPv4 service to set up the Client and set the IPv4 SNTP server. */
    status = nx_sntp_client_initialize_unicast(&demo_client, SERVER_IP_ADDRESS);


#else   /* Broadcast mode */

    /* Initialize the Client for broadcast mode, no roundtrip calculation required and a broadcast SNTP service. */

    /* Use the IPv4 service to initialize the Client and set IPv4 SNTP broadcast address. */
    status = nx_sntp_client_initialize_broadcast(&demo_client,  NX_NULL, SERVER_IP_ADDRESS);
#endif  /* USE_UNICAST */


    /* Check for error. */
    if (status != NX_SUCCESS)
    {

        return;
    }

    /* 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 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). */

    base_seconds =  0xd2c96b90;  /* Jan 24, 2012 UTC */
    base_fraction = 0xa132db1e;

    /* Apply to the SNTP Client local time.  */
    status = nx_sntp_client_set_local_time(&demo_client, base_seconds, base_fraction);

    /* Check for error. */
    if (status != NX_SUCCESS)
    {

        return;
    }

    /* Run whichever service the client is configured for. */
#ifdef USE_UNICAST

    status = nx_sntp_client_run_unicast(&demo_client);
#else
    status = nx_sntp_client_run_broadcast(&demo_client);
#endif  /* USE_UNICAST */

    if (status != NX_SUCCESS)
    {
        return;
    }

    spin = NX_TRUE;

    /* Now check periodically for time changes. */
    while(spin)
    {

        /* First verify we have a valid SNTP service running. */
        status = nx_sntp_client_receiving_updates(&demo_client, &server_status);


        if ((status == NX_SUCCESS) && (server_status == NX_TRUE))
        {

            /* Server status is good. Now get the Client local time. */

            /* Display the local time in years, months, date format.  */
            status = nx_sntp_client_get_local_time(&demo_client, &seconds, &milliseconds, &buffer[0]);

            if (status == NX_SUCCESS)
            {
                printf("Date: %s\n", &buffer[0]);
            }

            /* Wait a while before the next update. */
            tx_thread_sleep(300);

            memset(&buffer[0], 0, BUFSIZE);
        }
        else
        {

            /* Wait a short bit to check again. */
            tx_thread_sleep(100);
        }
    }

    /* We can stop the SNTP service if for example we think the SNTP service has stopped. */
    status = nx_sntp_client_stop(&demo_client);

    if (status != NX_SUCCESS)
    {
        return;;
    }

    /* Set up another server and reinitialize the SNTP Client. */
#ifdef USE_UNICAST
    /* Initialize the Client for unicast mode to poll the SNTP server once an hour. */
    status = nx_sntp_client_initialize_unicast(&demo_client, SERVER_IP_ADDRESS_2);

    /* Check for error. */
    if (status != NX_SUCCESS)
    {
        return;
    }

    /* Now start the SNTP Client task back up. */
    status = nx_sntp_client_run_unicast(&demo_client);

    if (status != NX_SUCCESS)
    {
        return;

    }
#else    /* Start Client in broadcast */

    status = nx_sntp_client_initialize_broadcast(&demo_client, NX_NULL, SERVER_IP_ADDRESS_2);

    if (status != NX_SUCCESS)
    {
        return;
    }

    /* Now start the SNTP Client task back up. */
    status = nx_sntp_client_run_broadcast(&demo_client);

    /* Check for error. */
    if (status != NX_SUCCESS)
    {
        return;
    }
#endif

    spin = NX_TRUE;

    /* Now check periodically for time changes. */
    while(spin)
    {

        /* First verify we have a valid SNTP service running. */
        status = nx_sntp_client_receiving_updates(&demo_client, &server_status);

        if ((status == NX_SUCCESS) && (server_status == NX_TRUE))
        {

            /* Server status is good. Now retrieve the Client local time. */

            /* Display the local time in years, months, date format.  */
            status = nx_sntp_client_get_local_time(&demo_client, &seconds, &milliseconds, &buffer[0]);
            if (status == NX_SUCCESS)
            {
                printf("Date: %s\n", &buffer[0]);
            }

            /* It will be a bit longer till the next update. */
            tx_thread_sleep(200);

            memset(&buffer[0], 0, BUFSIZE);
        }

        /* Wait a short bit and try again. */
        tx_thread_sleep(100);
    }

    /* To return resources to NetX and ThreadX stop the SNTP client and delete the client instance. */
    status = nx_sntp_client_delete(&demo_client);

    return;
}


/* This application defined handler for handling an impending leap second is not
   required by the SNTP Client. The default handler below only logs the event for
   every time stamp received with the leap indicator set.  */

UINT leap_second_handler(NX_SNTP_CLIENT *client_ptr, UINT leap_indicator)
{

    /* Handle the leap second handler... */

    return NX_SUCCESS;
}

/* This application defined handler for handling a Kiss of Death packet is not
   required by the SNTP Client. A KOD handler should determine
   if the Client task should continue vs. abort sending/receiving time data
   from its current time server, and if aborting if it should remove
   the server from its active server list.

   Note that the KOD list of codes is subject to change. The list
   below is current at the time of this software release. */

UINT kiss_of_death_handler(NX_SNTP_CLIENT *client_ptr, UINT KOD_code)
{

UINT    remove_server_from_list = NX_FALSE;
UINT    status = NX_SUCCESS;

```

```

    /* Handle kiss of death by code group. */
    switch (KOD_code)
    {

        case NX_SNTP_KOD_RATE:
        case NX_SNTP_KOD_NOT_INIT:
        case NX_SNTP_KOD_STEP:

            /* Find another server while this one is temporarily out of service.  */
            status =  NX_SNTP_KOD_SERVER_NOT_AVAILABLE;

        break;

        case NX_SNTP_KOD_AUTH_FAIL:
        case NX_SNTP_KOD_NO_KEY:
        case NX_SNTP_KOD_CRYP_FAIL:

            /* These indicate the server will not service client with time updates
               without successful authentication. */


            remove_server_from_list =  NX_TRUE;

        break;


        default:

            /* All other codes. Remove server before resuming time updates. */

            remove_server_from_list =  NX_TRUE;
        break;
    }

    /* Removing the server from the active server list? */
    if (remove_server_from_list)
    {

        /* Let the caller know it has to bail on this server before resuming service. */
        status = NX_SNTP_KOD_REMOVE_SERVER;
    }

    return status;
}
```

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.

NX_SNTP_CLIENT_SLEEP_INTERVAL

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.

NX_SNTP_CURRENT_YEAR

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.

NTP_SECONDS_AT_01011999

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_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_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

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	(0x07) Invalid pointer input
NX_INVALID_INTERFACE	(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

client_ptr	Pointer to SNTP Client control block

Return Values

NX_SUCCESS	(0x00) Successful Client creation
NX_CALLER_ERROR	(0x11) Invalid caller of service
NX_PTR_ERROR	(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

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_CALLER_ERROR	(0x11) Invalid caller of service
NX_PTR_ERROR	(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.  */
```

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 the 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)	Successful Client Creation
NX_INVALID_PARAMETERS	(0x4D)	Invalid non pointer input
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of service

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.  */
```

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 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	(0x4D)	Invalid non pointer input
NX_PTR_ERROR	(0x07)	Invalid pointer input
NX_CALLER_ERROR	(0x11)	Invalid caller of service

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.  */
```

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

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 received update status
NX_PTR_ERROR	(0x07) Invalid pointer input

Allowed From

Initialization**,** Threads

Example

```
/* Determine if the SNTP Client is receiving valid udpates.  */
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

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

Return Values

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

Allowed From

Threads

Example

```
/* Determine if the SNTP Client is receiving valid udpates.  */
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 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 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_SNTP_CLIENT_ALREADY_STARTED	
	(0xD0C) Client already started
NX_SNTP_CLIENT_NOT_INITIALIZED	
	(0xD01) Client not initialized
NX_PTR_ERROR	(0x07) Invalid pointer input
NX_CALLER_ERROR	(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

client_ptr	Pointer to SNTP Client control block.

Return Values

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

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.  */
```

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) Successfully set local time
NX_PTR_ERROR	(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

client_ptr	Pointer to SNTP Client control block
time_update_cb	Pointer to callback function

Return Values

NX_SUCCESS	(0x00) Successfully set callback
NX_PTR_ERROR	(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

client_ptr	Pointer to SNTP Client control block

Return Values

NX_SUCCESS	(0x00)	Successful stopped Client thread
NX_SNTP_CLIENT_NOT_STARTED	(0xD)B)	SNTP Client thread not started
NX_PTR_ERROR	(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

`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
status	--------	Internal error occurred
NX_SNTP_ERROR_CONVERTING_DATETIME	(0xD08)	NX_SNTP_CURRENT_YEAR not defined or no local client time established
NX_SNTP_INVALID_DATETIME_BUFFER	(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

milliseconds	Milliseconds to convert
fraction	Pointer to milliseconds 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: 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:

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 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